

Modeling and Simulation Frameworks for Processing-in-Memory Architectures

Mahdi Aghaei^a, Saba Ebrahimi^a, Mohammad Saleh Arafati^a, Elham Cheshmikhani^a, Dara Rahmati^a, Saeid Gorgin^b, Jungrae Kim^b

^a*Dept. of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran*

^b*Dept. of Electrical and Computer Engineering, Sungkyunkwan University, Seoul, Korea*

Abstract

Processing-in-Memory (PIM) has emerged as a promising computing paradigm to address the memory wall and the fundamental bottleneck of the von Neumann architecture by reducing costly data movement between memory and processing units. As with any engineering challenge, identifying the most effective solutions requires thorough exploration of diverse architectural proposals, device technologies, and application domains. In this context, simulation plays a critical role in enabling researchers to evaluate, compare, and refine PIM designs prior to fabrication. Over the past decade, a variety of PIM simulators have been introduced, spanning low-level device models, architectural frameworks, and application-oriented environments. These tools differ significantly in fidelity, scalability, supported memory/compute technologies, and benchmark compatibility. Understanding these trade-offs is essential for researchers to select appropriate simulators that accurately map and validate their research efforts.

This chapter provides a comprehensive overview of PIM simulation methodologies and tools. We categorize simulators according to abstraction levels, design objectives, and evaluation metrics, highlighting representative examples. To improve accessibility, some content may appear in multiple contexts to guide readers with different backgrounds. We also survey benchmark suites commonly employed in PIM studies and discuss open challenges in simulation methodology, paving the way for more reliable, scalable, and efficient PIM modeling.

Keywords: Processing-in-Memory, Simulation and Modeling, Memory Simulators, Emerging Memories.

1. Introduction

1.1. The Necessity and Role of Simulation

Simulation forms the methodological backbone of modern computer architecture research. The prohibitive cost, design complexity, and risks associated with fabricating experimental hardware prototypes render physical evaluation impractical for most exploratory concepts. Simulation environments provide a controllable and reproducible framework to estimate performance, energy efficiency, and functional correctness before committing designs to silicon. Through simulation, designers can iterate rapidly on design parameters, identify performance bottlenecks, and evaluate competing architectural alternatives.

In the broader context of computing systems, reliance on simulation has grown in proportion to the complexity of hardware–software interactions. From instruction set simulators to detailed timing models, researchers employ a variety of tools to capture system behavior across different abstraction layers. These simulators help bridge the gap between high-level algorithmic modeling and low-level hardware verification enabling quantification of trade-offs among performance, area, power, and thermal metrics, well in advance of fabrication.

For emerging paradigms like Processing-in-Memory (PIM), simulation plays an even more essential role. Unlike conventional CPUs and GPUs, where architectural components are well established, PIM architectures are highly diverse and continuously evolving. Each design may integrate computation with memory arrays, logic layers adjacent to DRAM stacks, or specialized non-volatile devices. Constructing physical prototypes for every such configuration is economically unfeasible and technologically constrained. Consequently, simulation becomes the de facto tool for evaluating architectural ideas and quantifying their impact on system performance.

Moreover, simulation fosters scientific reproducibility. Published PIM architectures are often evaluated under varying workloads, device technologies, and memory hierarchies. Without standardized simulation methodologies, comparing results across studies is unreliable. Simulator frameworks provide a common foundation for validation, ensuring that future work can be consistently benchmarked prior research.

From an educational standpoint, simulation also democratizes architectural research. Graduate students and engineers can explore advanced design

ideas using open-source simulators without access to costly fabrication facilities. In the industry, simulators shorten design cycles by enabling hardware teams to evaluate multiple design options in parallel. Indeed, simulation is not merely a convenience, it is a prerequisite for innovation in data-centric computing.

1.2. Simulation in the Context of PIM

Processing-in-Memory (PIM) challenges the long-standing von Neumann paradigm by integrating memory and computation into a single substrate. This fusion significantly reduces data movement, the primary source of energy consumption and latency in modern workloads. However, this same integration complicates analysis. Unlike conventional CPU pipelines or standard DRAM interfaces, PIM involves simultaneous modeling of both computation and memory behaviors. Simulation therefore becomes the only practical approach for evaluating performance interactions and verifying architectural correctness.

Effective PIM simulation requires co-modeling across multiple subsystems, including the memory array, peripheral circuits, logic layer, and host interface. Each component introduces critical parameters, such as latency, energy consumption, bandwidth, and fault tolerance, that collectively shape system-level behavior. Moreover, the diversity of physical memory technologies (e.g., DRAM, SRAM, RRAM, PCM, MRAM, and hybrid 3D-stacked memories) demands flexible and extensible simulation frameworks that can adapt to diverse device characteristics.

Different abstraction levels serve different research purposes. Device-level simulators focus on electrical properties, such as resistive switching and retention. Circuit-level simulators analyze sense-amplifier timing behavior in components such as sense amplifiers, row activation, and bitline coupling. Architectural simulators estimate throughput and energy consumption across memory banks, channels, and embedded processing elements. Full-system simulators integrate CPUs, interconnects, and software stacks to provide end-to-end system insights. Together, these simulation layers enable researchers to traverse the entire design hierarchy, from device physics to system deployment, without requiring hardware.

Furthermore, PIM simulation enables algorithm–architecture co-design. Workloads such as machine-learning kernels, graph analytics, and database operations can be mapped to different PIM substrates to evaluate efficiency

gains. Feedback from simulation informs hardware designers about bottlenecks such as bandwidth limitations or thermal hotspots, prompting interactive architectural refinement. In this way, simulation functions as both a predictive and prescriptive instrument, guiding the evolution of PIM from concept to implementation.

1.3. A Timeline of PIM Simulators

The historical progression of PIM simulators mirrors the technological evolution of memory-centric computing. Early frameworks were largely DRAM-centric, extending memory simulators such as NVMain [1] with simplified compute modules. These tools captured bandwidth and latency but ignored computation–memory coupling.

Between 2016 and 2020, specialized simulators like PIMSim [2] and Ramulator-PIM [3] introduced programmable PIM kernels and near-memory logic models. From 2020 onward, research shifted toward architectural and full-system co-simulation, exemplified by PIMSimulator [4] and PiMulator [5].

Recent frameworks, including PIIMeval [6] and MultiPIM [7], integrate benchmark suites, performance counters, and support for AI/ML workloads. This progression highlights a steady movement from feasibility analysis to end-to-end performance evaluation and application-driven design. Figure 1 can illustrate this progression, showing the shift from memory-centric models to holistic, application-aware simulation frameworks.

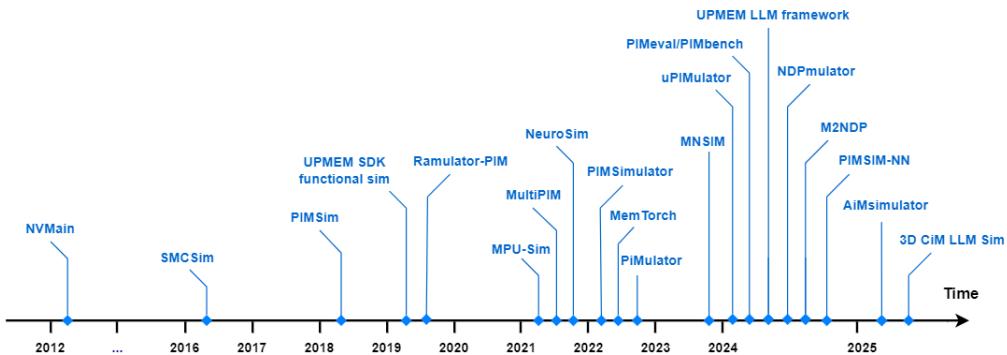


Figure 1: Timeline of PIM simulators

1.4. Practical Guide for Readers

Given the diversity of available tools, selecting the most suitable simulator depends on the reader’s specific objective and expertise. This section

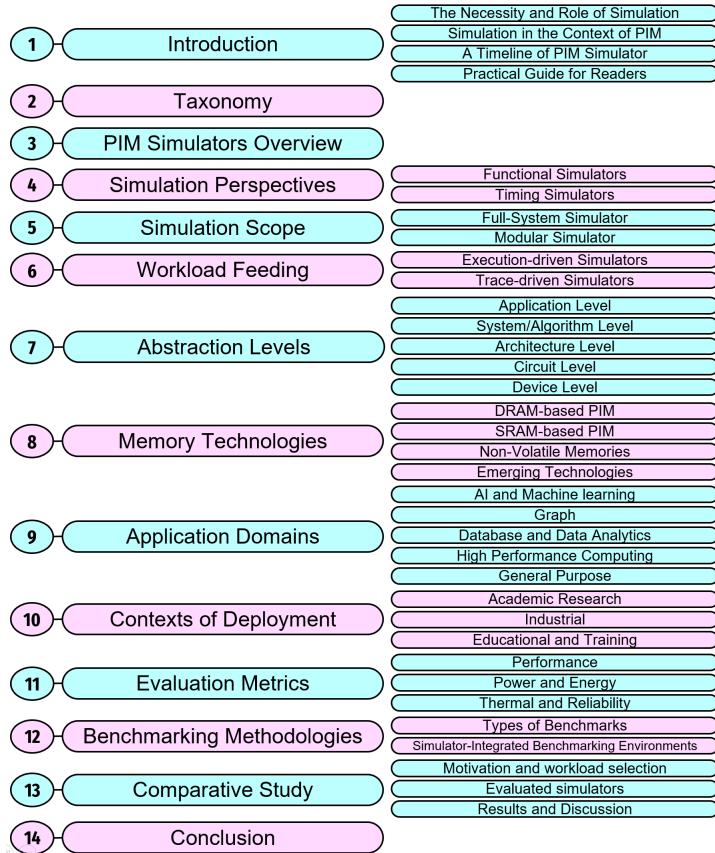


Figure 2: Overall structure of this chapter

provides a practical guide to help readers navigate the extensive and diverse landscape of PIM simulators. The organization of this chapter is intentionally hierarchical: early sections establish the conceptual foundation, while later sections delve into specialized simulation methodologies and case studies. The overview of the chapter's structure is depicted in Figure 2. Readers are encouraged to start with the initial taxonomy and overview sections before exploring more technical ones.

- Section 2 – Taxonomy of PIM Simulators: introduces a comprehensive classification framework used throughout the book. It defines key dimensions including accuracy, scope, abstraction, technology, and evaluation metrics, that underpin the organization of subsequent sections.

- Section 3 – Overview of Existing PIM Simulators: provides a concise survey of representative open-source simulators. It summarizes their design objectives, availability, and supported memory technologies, serving as a quick reference before readers engage with detailed methodological discussions.
- Sections 4 and 5 – Simulation Perspectives and Scope: together, these sections examine the methodological dimensions of PIM simulation. Section 4 discusses timing fidelity and functional accuracy, distinguishing between functional, event-driven, and cycle-level methodologies. Section 5 expands this discussion to system scope, comparing full-system versus modular simulation approaches and analyzing trade-offs between coverage, scalability, and integration complexity.
- Section 6 – Workload Feeding Mechanisms: details execution-driven and trace-driven methodologies used to provide stimuli to simulators.
- Section 7 and 8 – Abstraction Levels and Memory Technologies: discuss the range of simulation abstraction from device physics to system architecture, and examine how various memory technologies influence simulation design.
- Section 9 and 10 – Application Domains and Deployment Context: present how simulators target distinct workloads (AI, graph analytic, database processing, HPC) and cater to different user communities (academic, industrial, educational).
- Section 11 – Evaluation Metrics for PIM Simulation: synthesizes methodologies for assessing performance, energy, and reliability, and introduces benchmark suites such as PIMBench.
- Section 12 – Benchmarking Methodologies for PIM Architectures: introduces benchmark suites and standardized workloads designed specifically for PIM evaluation. It examines representative frameworks such as PIMBench, and their domain-tailored workloads for AI inference, graph traversal, and database acceleration.
- Section 13 – Comparative Study of PIM Simulators: presents a quantitative evaluation of two representative open-source PIM simulation

frameworks, PIMeval and PIMSimulator. It compares their modeling depth, configuration flexibility, and performance predictions across GEMM and GEMV workloads, highlighting trade-offs between functional abstraction and timing fidelity in PIM system evaluation.

- Section 14 – Conclusion: Summarizes key insights from all previous chapters, highlighting open challenges and future research directions in PIM simulation, including fidelity scaling, integration with machine-learning-driven design-space exploration, and co-verification with hardware prototypes.

2. Taxonomy of PIM Simulators

This section presents a compact taxonomy that organizes PIM simulators along eight orthogonal axes. Each axis captures a key design decision that researchers must consider when selecting simulation tools, and each is expanded in detail in later sections. The goal is to provide a concise, self-contained “map” that enables readers to navigate directly to the most relevant deep-dive with a shared terminology.

2.1. Accuracy Perspectives

PIM simulators differ fundamentally in how they model time. Functional simulators prioritize logical correctness and rapid iteration. They validate ISA extensions, kernel semantics, and software stacks behavior without incorporating detailed timing [8, 9]. Timing simulators including models, capture dynamic behaviors such as queuing, hazards, contention, and device latencies. These are essential for quantifying execution throughput, latency, and overlap between host and PIM execution [9, 10]. In practice, researchers often follow a top-down workflow: start with functional models, for rapid prototyping and progressing to timing-accurate simulations to refine insights. Hybrid models augment functional cores with analytical delay and energy estimators, providing early visibility into performance trends while maintaining simulation speed. Readers seeking trade-offs between speed and fidelity, and the distinctions among functional, event-driven, and cycle-accurate approaches, should see Section 4, where these perspectives are decomposed, exemplified, and compared with representative tools.

2.2. Scope and Granularity

Scope describes *how much of the system* a simulator models. *Full-system* frameworks emulate CPUs, memory, I/O, and OS, enabling end-to-end studies of scheduling, drivers, coherence, and virtual memory interaction with PIM [11]. Modular frameworks focus on specific subsystems (e.g., DRAM timing, vault controllers, or PIM kernels) for scalability and targeted microarchitectural analysis [9]. Granularity also plays a role: coarse-grain models accelerate design-space sweeps by simplifying details; fine-grain ones expose bottlenecks and corner cases but are typically slower. When selecting scope and granularity, align them with your question: Prefer modular models for microarchitectural sensitivity analysis or “what-if” explorations of individual design components. A structured discussion and case study appear in Section 5.

2.3. Workload Feeding Mechanisms

How a simulator is “fed” determines realism, reproducibility, and cost. *Execution-driven* flows run binaries inside the simulated platform, naturally exposing dynamic effects (speculation execution, runtime scheduling, contention) and eliminating massive trace files [11]. *Trace-driven* simulators flow replay pre-collected instruction/memory traces, trading some dynamism for excellent repeatability and fast multi-configuration sweeps; they are ideal for broad design exploration and regression testing [9]. Many projects combine both approaches: generate traces with an execution-driven run, then replay across architectures or device models. For a systematic comparison, practical guidelines, and representative tools, see Section 6.

2.4. Abstraction Levels

PIM research spans five principal abstractions: *Application* (end-to-end workload behavior and KPIs), *System/Algorithm* (mapping/scheduling across host and memory-side compute), *Architecture* (controllers, kernels, interconnects, coherence), *RTL/Circuit* (cycle-level pipelines, per-block timing/power), and *Device* (cell physics, endurance, variability) [12]. Choosing the right level hinges on your hypothesis and validation needs. For example, algorithm fit and speedups suggest application/system levels; ISA/coherence proposals require architectural timing; reliability/analog effects push toward circuit/device co-simulation. Cross-level coupling (e.g., architecture+device) is increasingly common for non-idealities and thermal constraints. A detailed, level-by-level treatment appears in Section 7.

2.5. Memory Technologies

The underlying memory technology shapes the set of feasible PIM operations, timing characteristics, and energy consumption profiles. DRAM-based PIM solutions (DDR/HBM/HMC/UPMEM) favors near-bank digital kernels and offers high internal bandwidth; SRAM-based designs trade density for ultra-low latency and tight on-chip integration; Non-volatile options (ReRAM/PCM/MRAM/FeFET) enable analog/digital in-place compute with endurance and variability trade-offs [13, 14]; emerging stacks and coherent fabrics (e.g., CXL-attached NDP) broaden integration possibilities. Selecting or abstracting the appropriate technology model is crucial to draw valid conclusions about performance, energy efficiency, and reliability. Technology-specific modeling considerations are comprehensively surveyed in Section 8.

2.6. Application Domains of PIM Simulators

PIM simulators target distinct workload classes with different access patterns and compute intensities. AI/ML emphasizes GEMM/GEMV, attention, and Key-Value (KV)-caches [15]; *Graph* processing workloads stresses irregular, pointer-chasing memory [16]; *Databases/Analytics* are dominated by scans, joins, and reductions [17]; *High-performance computing (HPC)* mixes dense/sparse linear algebra and stencil codes [18].

Some frameworks are *domain-agnostic*, offering configurable kernels and memory backends for broad cross-domain comparisons. Domain alignment is essential for credible evaluation: a simulator validated on CNN inference may not capture graph frontier behavior or OLAP joins. Representative mappings and decision criteria are provided in Section 9.

2.7. Contexts of Deployment

The same simulator can serve different proposes across communities. *Academic* settings use prioritizes extensibility, openness, and rapid prototyping for new ideas and cross-layer experiments. *Industrial* contexts emphasize calibrated timing, determinism, toolchain integration, and pre-silicon verification capabilities. *Educational/training* environments favor clarity, setup simplicity, and visual explainability. Recognizing these diverse deployment contexts helps set appropriate accuracy/runtime expectations and reporting standards (e.g., reproducible configs vs. silicon-correlated models). Guidance and representative tools tailored to each context appear in Section 10.

2.8. Evaluation Metrics for PIM Simulation

Robust quantitative assessment hinges on a balanced metric suite: *performance* (latency, throughput, bandwidth efficiency), *power/energy* (TOPS/W, total energy), and *thermal/reliability* (hotspots detection, fault injection, endurance/ retention) [11]. Cross-simulator comparability requires transparent methodology including identical datasets, calibration points, warm-up/ measurement intervals, and statistical treatment. A practical approach combines fast analytical models for sweeps with detailed runs for focal points, and report sensitivity to technology and mapping choices. A structured treatment, including cross-validation practices, is provided in Section 11.

2.9. PIM-Specific Benchmarks

Benchmarks operationalize the taxonomy by standardizing inputs, kernels, and evaluation metrics. General benchmark suites (e.g., ML/DL, graph, database, and HPC) should be complemented with *PIM-specific* collections that expose near-memory kernels (reductions, scans, atomics, bitwise ops, analog MVM) and memory-stressing access patterns.

High-quality PIM benchmarks define clear input scaling rules, mapping rules, and reference baselines (e.g., CPU, GPU, or accelerator), along with harnesses capable of generating traces or executing natively on simulators. They should also document hardware and technology assumptions to ensure fair comparisons in latency and energy efficiency. A survey of available benchmark suites and their workload coverage are presented in Section 12.

3. PIM Simulators Overview

This section provides a comprehensive overview of the publicly available Processing-in-Memory (PIM) simulators that have been released to the research community. The focus is deliberately restricted to frameworks whose source code is accessible, either through open-source repositories or publicly shared academic distributions, ensuring that the tools discussed here can be directly used, verified, and extended by other researchers. While numerous PIM-related simulators have been proposed over the past decade, many remain proprietary, inaccessible, or documented only at a high level. Such tools, though valuable for industrial or internal evaluation, cannot serve as reproducible baselines for academic investigation. Therefore, only simulators that are fully or partially open-source, accompanied by functional documentation or repositories, are included in this section.

3.1. NVMain

NVMain [1] is a cycle-accurate, architectural-level main memory simulator developed to model both conventional DRAM and emerging non-volatile memory (NVM) technologies such as ReRAM, STT-RAM, and PCM. It was designed to address the limitations of DRAM-centric simulators (e.g., **DRAMSim**) that lack support for endurance modeling, asymmetric read/write latencies, and hybrid DRAM–NVM systems. The simulator models complete memory subsystems—including channels, ranks, banks, and subarrays—with fully customizable timing parameters (e.g., $tRCD$, $tRAS$, tWR , $tFAW$) and built-in verification for command legality and bus contention. Moreover, **NVMain** can import latency and energy parameters from circuit-level tools such as **NVSim** (built on **CACTI**), thereby bridging device-level modeling with architectural exploration. Its modular architecture supports experimentation with memory controllers, refresh policies, and interconnects—from traditional DDR buses to alternative optical or serial links.

In addition to timing and performance, **NVMain** incorporates detailed endurance and reliability modeling, enabling long-term studies of wear-out behavior and fault propagation in non-volatile arrays. It supports advanced techniques such as Data-Comparison Write (DCW) and Flip-N-Write to minimize redundant bit transitions, improving both write energy efficiency and device lifetime. Power estimation can rely on IDD-based DRAM power models or on technology-specific data imported from **NVSim/CACTI**, while endurance-aware policies and refresh strategies can be evaluated under realistic workload traces.

Due to its extensibility and cycle-level precision, **NVMain** has become a foundational component in subsequent PIM-related simulators such as **PIMSim**, **MultiPIM**, and **NDPmulator**. These frameworks reuse **NVMain**’s validated energy and timing models as their back-end memory substrate to support heterogeneous DRAM–NVM configurations and to analyze data-movement efficiency in near-memory computing systems. Overall, **NVMain** remains one of the most influential open-source simulators for bridging the gap between circuit-level device models and system-level architectural evaluation of next-generation memory technologies.

3.2. NVSim

NVSim [19] is a circuit-level modeling framework designed to estimate the area, latency, dynamic energy, and leakage power of both volatile and nonvolatile memory (NVM) technologies. Supporting a wide spectrum of

emerging memories—such as Phase-Change Memory (PCM/PCRAM), Spin-Transfer Torque RAM (STT-RAM/MRAM), Resistive RAM (ReRAM), Floating-Body DRAM (FBDRAM), and NAND Flash—**NVSim** provides a unified analytical platform for device-to-architecture performance evaluation.

Developed as an extension of the modeling concepts introduced by **CACTI**, **NVSim** emphasizes configurability and extensibility, offering detailed parameterization of banks, mats, subarrays, and peripheral circuits. Its framework accommodates diverse routing topologies (e.g., H-tree, bus-based architectures) and multiple sensing schemes—current-mode, voltage-mode, and voltage-divider—allowing accurate capture of technology-specific read and write behaviors. Furthermore, it integrates process-dependent parameters, such as resistance range, write current, and retention characteristics, to support exploration of novel NVM materials and device structures.

Validation results demonstrate that **NVSim** achieves high modeling fidelity, typically within 30% of empirical measurements from industrial prototypes. While originally developed for conventional memory design rather than Processing-in-Memory (PIM), **NVSim** has become a foundational component for higher-level PIM and Compute-in-Memory (CIM) simulators. Many later tools—such as **NVSim-PIM**, **NVMain**, and **NeuroSim**—inherit its timing and power estimation modules to enable more complex architectural and system-level modeling.

In essence, **NVSim** functions as a functional, modular, and circuit-level simulator that facilitates early-stage design-space exploration of emerging memory technologies. Although it does not provide full-system or cycle-accurate simulation, its ability to bridge device-level parameters with architectural-level insights has made it a cornerstone in the evaluation and optimization of both standalone NVM designs and memory-centric computing architectures.

3.3. *SMCSim*

SMCSim [20] is a full-system, gem5-based simulation framework developed to model and evaluate near-memory computation within the Smart Memory Cube (SMC)—an enhanced evolution of the Hybrid Memory Cube (HMC) that integrates programmable compute units in its logic base (LoB). Built atop gem5’s General Memory System infrastructure, **SMCSim** reuses fundamental components such as DMA engines, interconnect fabrics, and DRAM controllers, while augmenting them with serial link modeling and SMC-specific device logic to accurately represent the 3D-stacked memory organization and its embedded compute layer. The simulator provides a

timing-accurate environment for modeling memory and interconnect subsystems, alongside functional modeling of the host SoC and in-memory processing cores. It supports scratchpad memories, DMA-driven data transfers, and complete address translation mechanisms (TLB/MMU), accompanied by a software stack comprising device drivers and user-level APIs that enable transparent workload offloading while maintaining virtual memory semantics.

At the architectural level, **SMCSim** partitions the SMC into multiple DRAM vaults, each equipped with local controllers and compute resources connected through a configurable logic-base interconnect. The framework integrates DRAM timing and bandwidth models, interconnect latency, and address-mapping policies, facilitating detailed exploration of system-level trade-offs such as data placement, thread scheduling, and inter-vault communication efficiency in 3D-stacked near-memory systems. Through its extensible interface, users can configure vault count, link bandwidth, and interconnect topology, making the simulator adaptable to a wide range of experimental scenarios.

Overall, **SMCSim** provides a reproducible, timing-aware platform for investigating Smart Memory Cube-style PIM architectures. It effectively captures host–memory interactions, data movement patterns, and interconnect behavior while maintaining a significantly lower computational overhead compared to full RTL models. Owing to this balance between accuracy and efficiency, **SMCSim** serves as a practical research tool for architectural exploration, performance analysis, and hardware/software co-design in emerging near-memory processing systems.

3.4. *PIMSim*

PIMSim [2] is a comprehensive full-system simulation framework that supports a wide range of Processing-in-Memory (PIM) architectures. It addresses several limitations of earlier tools—such as fragmented toolchains and rigid hardware assumptions—by integrating multiple timing back-ends including **DRAMSim2**, **HMCSim**, and **NVMain**. Through this unified backend infrastructure, **PIMSim** enables co-simulation of hybrid memory systems such as DRAM, HMC, and non-volatile memories under a consistent timing and coherence model, thereby facilitating heterogeneous PIM system exploration within a single environment.

The simulator operates in three distinct modes that balance fidelity and performance according to design objectives. The first mode, known as *Fast*

Mode, targets rapid architectural prototyping. It employs pre-generated memory traces and simplified pipeline models to estimate performance and energy without modeling OS-level overheads, allowing researchers to test new logic designs or scheduling strategies within seconds. The second mode, *Instrumentation-Driven Mode*, bridges trace-driven and full-system simulations. Leveraging Intel Pin as a front-end, it dynamically instruments running applications, identifies annotated PIM kernels at runtime, and redirects them to the simulator for online evaluation—offering realistic workload partitioning at moderate simulation cost. Finally, the *Full-System Mode* extends gem5 with PIM-specific instructions, detailed coherence protocols, and full system-call handling. It supports multiple ISAs (x86, ARM, SPARC), integrates fine-grained coherence mechanisms such as MESI and LazyPIM, and allows users to instantiate PIM logic as CPU-like cores, GPU units (via GPGPU-Sim), or abstract accelerators.

This hierarchical design enables researchers to transition seamlessly between fast design-space exploration and cycle-accurate full-system analysis within the same framework. By combining modular timing back-ends, flexible front-end interfaces, and broad architectural coverage, PIMSim provides a versatile and extensible foundation for investigating heterogeneous memory systems, host–PIM coherence, and near-data computation with consistent architectural semantics and reproducible results.

3.5. UPMEM Functional Simulator

UPMEM Functional Simulator [21] is an integral component of the official UPMEM SDK, developed to emulate the behavior of UPMEM’s Data Processing Units (DPUs) in the absence of physical hardware. The simulator is automatically activated when no DPU-equipped DIMM is detected in the system, though users can also enable it explicitly through the SDK environment configuration. It creates virtual ranks composed of one simulated chip per rank and supports up to 64 DPUs, maintaining compatibility with the same software stack used on real UPMEM hardware.

The simulator provides functional correctness rather than cycle accuracy. It executes compiled DPU binaries, emulating instruction behavior and host–DPU communication, but does not model timing, latency, or energy consumption. This design makes it highly suitable for application development, debugging, and verification of PIM programs prior to deployment on physical UPMEM modules. Since it faithfully reproduces the logical execution flow and data exchanges between the host and DPUs, the

simulator operates across multiple abstraction levels, including application, system/algorithm, and architectural layers.

Overall, the UPMEM Functional Simulator offers a convenient and accessible software environment for developing, validating, and profiling near-memory computing workloads without requiring specialized hardware. It has become an essential tool for researchers and developers seeking to explore the UPMEM ecosystem and test large-scale memory-centric applications in a fully virtualized setup.

3.6. Ramulator-PIM

Ramulator-PIM [3] extends the widely adopted **Ramulator** DRAM simulator to incorporate cycle-accurate support for Processing-in-Memory (PIM) architectures. It retains Ramulator’s high-fidelity timing model while adding a dedicated flow that enables offloading of application kernels to simple in-order logic cores embedded within the 3D-stacked memory’s logic layer. The simulation operates in a trace-driven manner: a modified **ZSim** frontend executes the host program, producing two complementary traces—a filtered trace representing the CPU-side execution (capturing cache and coherence effects) and an unfiltered trace representing the PIM kernel execution. These traces are replayed within **Ramulator-PIM** to capture cycle-accurate memory timing, vault-level parallelism, and off-chip SerDes latencies. While the current public release does not support concurrent host–PIM execution, it serves as a precise framework for comparing kernel performance between host and PIM cores under realistic DRAM behavior.

To mitigate the inherently long runtime of full cycle-accurate simulations, NAPEL (Near-memory computing Application Performance & Energy prediction via ensemble Learning) [22] was introduced as a learning-based extension atop **Ramulator-PIM**. NAPEL leverages LLVM-based static analysis to extract a rich set of hardware-agnostic features from PIM kernels, such as instruction mix, memory reuse distance, and traffic patterns, and applies a Central Composite Design (CCD) methodology to select a minimal yet representative set of kernels and configurations for detailed simulation. The collected results are used to train a Random Forest model that accurately predicts performance (IPC) and energy for unseen kernels within the same near-memory configuration. Reported evaluations indicate prediction errors of approximately 8.5% for performance and 11.6% for energy, while achieving up to 220 \times faster design-space exploration compared to exhaustive simulation.

Ramulator-PIM employing NAPEL establishes a hybrid, calibration-driven workflow that combines the precision of cycle-accurate DRAM and logic-layer timing with the scalability of machine-learning-based performance prediction. This integration enables researchers to explore large PIM design spaces efficiently, maintaining analytical rigor without incurring the prohibitive cost of full-scale simulation across all workloads and configurations.

3.7. *MPU-Sim*

MPU-Sim [23] is a cycle-accurate, research-grade simulator designed for evaluating near-bank processing in 3D-stacked DRAM architectures. It models a SIMT-style processor located on the base logic die and lightweight near-bank units (NBUs) integrated within individual DRAM dies. CUDA programs are compiled to PTX and executed on simple in-order subcores within the logic die, with selective offloading of computation to NBUs positioned adjacent to DRAM bank groups. This design exploits high internal bank bandwidth and minimizes data movement between compute and memory layers.

The simulator takes three main inputs, application program, data placement, and thread-block scheduling, to facilitate detailed studies on compiler back-end offloading, memory layout, and runtime scheduling strategies. Compared with prior open-source PIM frameworks, **MPU-Sim** incorporates near-bank specific features such as per-bank control logic, TSV arbitration, and a split execution pipeline for offloading, while targeting general-purpose SIMT workloads rather than fixed-function PIM kernels.

MPU-Sim generates fine-grained profiling traces and detailed component-level statistics that feed into static and dynamic energy models. Its components are cross-validated against well-established simulators: **GPGPU-Sim** for SIMT core behavior, **DRAMSim2** for DRAM timing, and **BookSim** for NoC interconnect modeling. Case studies highlight its analytical power—for example, demonstrating how DRAM refresh policies can significantly influence near-bank performance, and how aligning thread-block scheduling with bank mapping can yield substantial throughput gains.

Distributed as an open-source framework with containerized (Docker) setup scripts, **MPU-Sim** provides a robust and reproducible environment for exploring compiler, runtime, and architectural co-design in bank-level PIM systems. It stands as one of the few publicly available simulators combining CUDA/PTX programmability with detailed DRAM and NoC modeling,

making it a valuable tool for studying the interaction between memory hierarchy organization and near-data compute efficiency.

3.8. MultiPIM

MultiPIM [7] is a detailed and highly configurable simulator for modeling large-scale Processing-in-Memory (PIM) systems built from multiple 3D-stacked memory devices. Unlike earlier single-stack frameworks such as **PIMSim** or **Ramulator-PIM**, it captures the realities of multi-stack interconnects, directory-based coherence, and virtual memory, all of which are essential to evaluate practical, scalable PIM designs.

Each memory stack is divided into several vaults, and every vault contains a lightweight in-order PIM core. Intra-stack connections, the local network that links vaults to the logic base of the same stack, are modeled as cycle-accurate crossbars and TSV links. Inter-stack connections, the high-speed links between different stacks, are configurable and simulated cycle-accurately using **BookSim2**. The simulator’s backend, built on **Ramulator**, models DRAM timing and interconnect delays, while the frontend, based on **ZSim**, executes host instructions, manages thread scheduling, and drives memory traffic. This two-layer architecture gives **MultiPIM** both scalability and cycle-level accuracy for complex multi-stack memory systems.

MultiPIM provides a source-level programming interface that allows developers to mark offloadable code regions with simple directives such as `pim_blk_begin/end()` or `pim_mp_begin/end()`. These APIs can automatically map POSIX/OpenMP threads or thread blocks onto the distributed in-memory PIM cores. The simulator supports user-defined interconnect topologies described via XML, and directory-based MESI coherence among PIM cores. It also integrates a unified virtual memory system, complete with TLBs and page table walkers, so both CPUs and PIM cores share a consistent address space. Profiling and statistics features help analyze latency, bandwidth bottlenecks, and coherence overheads in realistic applications.

3.9. NeuroSim for PIM

NeuroSim [24] is a functional, execution-driven, and modular simulation framework specifically developed for compute-in-memory (CIM) architectures employed in deep neural network (DNN) accelerators. It supports rapid and accurate design-space exploration across multiple levels of abstraction, from device physics to neural algorithm mapping, making it one of the most

extensively adopted frameworks for evaluating resistive-memory-based neural accelerators.

Unlike single-layer simulators, **NeuroSim** integrates a hierarchical analytical modeling structure that spans four abstraction levels. At the *device level*, it models memory cells and transistor technologies such as RRAM, PCM, and SRAM, incorporating non-ideal effects like variability and write asymmetry. At the *circuit level*, it captures the behavior of peripheral modules, including sense amplifiers, ADC/DAC converters, and array-level interconnects. At the *architecture level*, it organizes multiple subarrays into processing elements (PEs) and higher-order tiles interconnected through configurable routing networks. Finally, at the *algorithm level*, it supports mapping and execution of arbitrary neural network layers for both inference and training, providing an end-to-end link between network behavior and underlying memory hardware.

This hierarchical approach enables **NeuroSim** to estimate key design metrics such as area, latency, energy consumption, and accuracy degradation under realistic device conditions. Validation against post-layout SPICE simulations of a 40 nm RRAM-based CIM macro has demonstrated chip-level prediction errors below 1% after calibration, confirming its reliability for pre-silicon evaluation.

An extended version, **DNN+NeuroSim**, further integrates the hardware model with PyTorch-based DNN training and inference frameworks, offering full-stack co-simulation from device to algorithm. This integration allows users to perform cross-technology comparisons and assess trade-offs between energy efficiency, latency, and accuracy across both digital and analog memory-based computing paradigms.

Overall, **NeuroSim** stands as a comprehensive functional and analytical simulator that bridges device-level modeling with system-level neural network performance analysis. Its modularity, validated accuracy, and broad scope make it a cornerstone tool for the study and optimization of CIM and PIM architectures in next-generation AI accelerators.

3.10. *PIMSimulator*

PIMSimulator [4] is a cycle-accurate, DRAMSim2-backed memory simulator that models in-DRAM execution for HBM2-class systems. It augments a conventional DRAM timing substrate with a programmable SIMD PIM engine featuring scalar and vector register files (SRF/GRF) and a compact PIM ISA (e.g., ADD, MUL, MAC, MAD, MOV, FILL, JUMP). To coordinate near-data execution under realistic pseudo-channel timing, the simulator introduces

PIM-specific DRAM commands (`Activate_pim`, `ALU_pim`, `READ_pim`) that orchestrate data movement across banks, row buffers, and on-die compute units while preserving cycle-level fidelity.

By operating directly at the device/bank granularity, **PIMSimulator** enables quantitative studies of bank-level parallelism, in-memory data motion, and instruction-level kernel behavior without the confounding effects of full CPU/OS modeling. The resulting design space exploration remains lightweight yet timing-faithful for HBM-style PIM microarchitectures. In summary, **PIMSimulator** offers a concise, reproducible platform for prototyping and evaluating HBM-PIM kernels under realistic memory timing, complementing—but not replacing—full-system frameworks when OS, coherence, or multi-program effects are central to the research question.

3.11. *MemTorch*

MemTorch [25] is an open-source framework for simulating memristive (RRAM-based) deep learning systems, with a particular emphasis on modeling device-level non-idealities that affect neural network accuracy and reliability. Integrated natively with the PyTorch ecosystem, it enables users to train and evaluate conventional DNN and CNN models while transparently incorporating realistic hardware effects such as conductance drift, device variability, and limited precision into the computation flow.

MemTorch operates as a modular, execution-driven simulation environment. Instead of approximating abstract hardware performance, it executes PyTorch models directly through a simulation backend that intercepts layer operations at runtime to emulate in-memory multiply–accumulate (MAC) and convolutional operations within memristive crossbars. This allows the framework to capture the direct impact of device physics and circuit behavior on algorithmic outcomes during network execution and training.

The framework includes several physics-based device models—such as Linear Ion Drift, VTEAM, Stanford–PKU RRAM, and data-driven Verilog-A implementations—that describe conductance evolution, state transitions, and endurance degradation in resistive memory devices. By co-simulating both device-level non-idealities and peripheral circuit effects (including sensing, ADC/DAC quantization, and interconnect parasitics), **MemTorch** offers a highly realistic environment for studying how imperfections in emerging non-volatile memories influence the accuracy, robustness, and energy efficiency of neural networks.

MemTorch serves as a functional, modular, and execution-driven PIM-oriented simulator that spans multiple abstraction levels—from device and circuit modeling to algorithm and system integration. It provides researchers with a unified platform to investigate the effects of resistive memory variability and non-idealities on deep learning performance, enabling accurate hardware–software co-design and evaluation for next-generation compute-in-memory neural accelerators.

3.12. *PiMulator*

PiMulator [5] is a modular and parameterizable FPGA-based platform built to accelerate the prototyping and evaluation of Processing-in-Memory (PIM) architectures with high fidelity. Implemented entirely in SystemVerilog and integrated within the LiteX SoC builder framework, **PiMulator** enables flexible configuration of both memory hierarchies and PIM compute logic. Researchers can instantiate custom architectures directly on FPGA hardware, adjusting timing, bandwidth, and interconnect parameters to reflect a wide range of memory technologies and design choices.

Operating across circuit- and architecture-level abstractions, **PiMulator** bridges low-level DRAM behavior with higher-level system evaluation. The framework models DDR4 and HBM memory subsystems, including channels, ranks, banks, and subarrays, while capturing timing-critical behaviors such as row activation, refresh, and inter-bank data movement. It further supports emulation of well-known DRAM-PIM mechanisms such as RowClone, Ambit, and LISA, facilitating direct comparison between compute- and data-movement-oriented approaches. With its event-driven hardware emulation pipeline, **PiMulator** achieves up to 28 \times runtime speedup compared to traditional software simulators, enabling large-scale, near-real-time exploration of complex PIM workloads.

Unlike software-only tools, **PiMulator** runs on FPGA fabric, providing cycle-faithful evaluation while maintaining configurability and modularity. Users can extend the base framework to include new compute primitives, interconnect topologies, or even non-volatile memory models through modular HDL components. This versatility makes it suitable for both architectural exploration and hardware/software co-design of PIM systems.

In summary, **PiMulator** is a functional, execution-driven, and modular hardware-accelerated simulation framework that offers researchers a bridge between simulation and silicon. By combining reconfigurability, speed, and fidelity, it serves as a practical open-source environment for evaluating DRAM-

and HBM-based PIM designs, fostering rapid iteration from concept to prototype.

3.13. MNSIM 2.0

MNSIM 2.0 [26] is a behavior-level simulation framework developed to evaluate the performance, energy efficiency, and computational accuracy of Processing-in-Memory (PIM) architectures, particularly those used in AI acceleration. It addresses the growing need for rapid and accurate evaluation of memory-centric computing by eliminating the reliance on time-consuming circuit-level simulations. The tool provides a flexible, modular modeling environment that allows users to configure both digital and analog PIM arrays, peripheral circuits, and interconnect architectures, enabling comprehensive system-level performance assessment across large-scale neural workloads.

A major innovation of MNSIM 2.0 lies in its unified array modeling structure, which supports heterogeneous memory technologies—such as RRAM-based analog crossbars and SRAM-based digital PIM units—under a single framework. This unified abstraction allows for consistent comparison between different compute-in-memory designs and facilitates hybrid evaluations where analog and digital subarrays coexist. The simulator integrates a PIM-oriented neural network training and quantization pipeline that co-optimizes neural models and hardware parameters, capturing device-level non-idealities, quantization noise, and limited precision effects during model development. Additionally, its scheduling interface enables flexible mapping of neural layers to hardware resources, supporting diverse dataflows (e.g., output-stationary, weight-stationary) and pipelined execution modes.

Operating at the behavioral level, MNSIM 2.0 delivers results in seconds while maintaining high predictive accuracy, with reported deviation within 3.8%–5.5% compared to post-layout measurements of fabricated PIM macros. The framework provides detailed breakdowns of latency, power, energy, and accuracy degradation, helping designers identify performance bottlenecks and optimization opportunities. Its extensibility also allows integration with higher-level design frameworks and external deep-learning toolchains, making it suitable for end-to-end AI hardware–software co-design.

MNSIM 2.0 is a functional, modular, and execution-driven simulation environment spanning architecture, system/algorithm, and application abstraction levels. It offers a fast yet accurate platform for evaluating PIM architectures in neural accelerators and supports comprehensive co-design exploration, bridging the gap between device behavior, architectural efficiency,

and algorithmic robustness in next-generation AI computing systems.

3.14. *uPIMulator*

uPIMulator [27] is a cycle-accurate simulator designed to execute real UPMEM binaries. It relies on the official UPMEM LLVM toolchain to compile C kernels, which are then run on a detailed model of the DPU (an in-DRAM core) which closely interacts with its DDR4 memory bank. The simulator models the 14-stage in-order pipeline, “revolver” thread scheduling, register hazards, and on-chip memories (WRAM, IRAM, MRAM). Host–device data transfers are characterized using asymmetric DMA bandwidths and realistic pseudo-channel timing, enabling researchers to investigate how kernels interact with both compute and memory resources under real hardware-like limitations.

uPIMulator [27] is a cycle-accurate simulator calibrated for real UPMEM binaries. It uses the official UPMEM LLVM toolchain to compile C kernels and executes them on a detailed model of the DPU (an in-DRAM core) closely coupled to DDR4 memory banks, modeling a 14-stage in-order pipeline, “revolver” thread scheduling, register hazards, and on-chip memories (WRAM/IRAM/MRAM). It executes actual DPU binaries and characterizes host–device transfers with asymmetric DMA bandwidth and realistic pseudo-channel timing, enabling analysis of compute–memory interactions under hardware-like constraints.

Beyond ensuring functional correctness, **uPIMulator** is designed for microarchitectural exploration. Its modular front-end and back-end allow researchers to prototype SIMT/vector execution, apply instruction-level parallelism (ILP) techniques such as superscalar issue and data forwarding, and experiment with memory system modifications like replacing scratchpads with caches. It also supports virtual memory through a lightweight MMU and TLB, enabling studies on multi-tenant isolation and pointer safety with only 0.8% performance overhead in published evaluations.

Overall, **uPIMulator** provides a practical, hardware-faithful platform for evaluating UPMEM-style PIM. It runs real ISA code on a configurable microarchitecture, making it ideal for analyzing compute–memory bottlenecks and testing architectural extensions such as vector units, ILP, virtual memory, and cache-based designs before hardware implementation.

3.15. PIMeval

PIMeval [6] is a lightweight, functional simulation framework designed to evaluate the performance and energy efficiency of digital DRAM-based Processing-in-Memory (PIM) architectures. Together with its companion benchmark suite, **PIMbench**, it provides a unified evaluation platform that allows fair and reproducible comparison across different PIM organizations. The framework supports multiple PIM design styles, including bit-serial subarray PIM, bit-parallel subarray (Fulcrum-style) PIM , and bank-level PIM. Through a unified PIM API, the toolchain abstracts low-level hardware differences, mapping high-level operations such as arithmetic, logical, and reduction functions into corresponding micro-operations specific to each PIM architecture. This abstraction layer enables seamless cross-platform evaluation, ensuring that applications written once can be executed and analyzed across multiple architectural variants.

Internally, **PIMeval** models performance and energy by dividing execution into two principal phases—data movement and computation. Timing analysis incorporates DRAM activation, precharge, and data-transfer latencies, while energy estimation combines standard DDR power models with additional contributions from on-die compute logic and interconnect circuitry. Although the current version relies on analytical models for latency estimation, future integration with **DRAMSim3** is planned to provide cycle-level timing accuracy. This makes **PIMeval** particularly suitable for early-stage design exploration and comparative analysis across digital PIM architectures.

The accompanying benchmark suite, **PIMbench**, provides a diverse set of workloads representative of key PIM application domains, including vector arithmetic, matrix operations, graph analytics, and machine learning kernels. Each workload is implemented once using the unified PIM API, ensuring portability and fair comparison across architectures. Together, **PIMeval** and **PIMbench** form a coherent ecosystem for PIM evaluation, bridging the gap between architectural modeling and application-level benchmarking, and enabling researchers to quantify performance and energy trade-offs in a consistent and reproducible manner.

3.16. UPMEM LLM framework

UPMEM LLM framework [28] is a functional simulation framework developed as part of UPMEM’s PIM-AI architecture, designed to accelerate large-scale language model (LLM) inference by integrating compute logic directly within DDR5 and LPDDR5 memory devices. Each PIM-AI chip can operate

in two distinct modes: a *Non-PIM mode*, functioning as a conventional 2 GB DRAM device, and a *PIM mode*, where it acts as a neural accelerator capable of executing core transformer operations within the memory subsystem itself. This dual-mode capability enables flexible deployment in both conventional memory hierarchies and heterogeneous AI accelerators.

The accompanying simulator provides a high-level functional modeling environment for evaluating LLM inference across various memory-centric architectures. It runs unmodified PyTorch models, enabling users to execute transformer-based workloads directly without conversion or retraining. The framework offers configurable hardware profiles representing different accelerator types (e.g., GPU, NPU, or custom PIM configurations) and collects performance statistics such as execution latency, data-transfer volume, energy consumption, and power. It models both the compute-bound encoding phase (typically dominated by GEMM operations) and the memory-bound decoding phase (dominated by GEMV operations), while also accounting for KV-cache movement and inter-layer dependencies. Additionally, the simulator supports flexible layer-to-hardware mapping, allowing hybrid execution across host and memory-resident compute elements.

Architecturally, the simulator adopts a modular, execution-driven design that captures the functional behavior of heterogeneous components without cycle-level timing or operating system emulation. This abstraction allows rapid design exploration, high-level profiling, and scalability studies of memory-centric inference systems. While not cycle-accurate, its high-level modeling fidelity provides valuable insight into performance and energy trade-offs in large-scale LLM deployments on PIM-enabled platforms.

PIM-AI offers researchers a practical environment for studying GPU-free inference acceleration through tightly integrated compute–memory architectures. By combining realistic workload execution with flexible configuration options, it serves as an effective prototyping and analysis tool for next-generation, memory-centric AI systems.

3.17. *NDPmulator*

NDPmulator [29] is a gem5-based, cycle-accurate full-system simulator designed for detailed modeling and evaluation of Near-Data Accelerators (NDAcc) placed at various levels of the memory hierarchy—from caches to DRAM. It captures the complete CPU–memory–accelerator interaction, incorporating operating-system overheads, device-driver communication, and

multi-process contention, thus enabling realistic performance and energy evaluation across diverse system configurations.

The simulator enhances `gem5` with two major extensions. The first is a *Programming Interface (PI)* that abstracts CPU–accelerator coordination, managing synchronization, data transfers, and task invocation. The second is a *Load/Store Unit (LSU)* that decomposes large accelerator memory transactions into smaller, fine-grained accesses, preserving coherence and timing fidelity while allowing NDAccs to be integrated seamlessly at any cache or memory level. Moreover, `NDPmulator` supports virtual memory translation and address remapping, ensuring transparent and consistent data sharing between CPU and accelerator domains.

The framework supports two complementary operation modes: (i) *System Emulation (SE)*, which enables fast simulation of single-application workloads without OS overheads—ideal for early-stage microarchitectural exploration; and (ii) *Full-System (FS)*, which runs a complete Linux stack, capturing driver latency, kernel scheduling, and resource contention for system-level accuracy. This dual-mode design allows researchers to trade off speed and fidelity depending on their study objectives.

Overall, `NDPmulator` provides a unified simulation infrastructure that bridges rapid design-space exploration with full-system realism, enabling comprehensive analysis of near-data accelerators and their interactions with modern memory hierarchies.

3.18. M^2NDP

M^2NDP [30] is a modular simulation framework designed to evaluate Near-Data Processing (NDP) architectures for Compute Express Link (CXL)-based memory systems. It introduces a Memory-Mapped NDP model that enables computation to occur directly within the CXL memory controller, thereby minimizing data movement and latency compared to conventional host-driven processing. The framework is built around two key architectural primitives: M^2Func , a `CXL.mem`-compatible communication interface between the host processor and the NDP controller, and $M^2\mu Thread$, a lightweight microthreading engine designed to support fine-grained concurrency with minimal software overhead. Together, these mechanisms enable substantial performance and energy improvements—reportedly achieving up to 128 \times higher performance and 87.9% lower energy consumption than CPU/GPU hosts equipped with passive CXL memory devices.

Constructed atop the `Ramulator` memory simulation core, M^2NDP integrates both functional and timing simulation capabilities, allowing users to model execution correctness while also capturing detailed latency, bandwidth, and power characteristics. The simulator adopts a modular design rather than a full-system one, making it particularly suitable for design-space exploration of individual PIM or NDP components, while remaining flexible for integration with higher-level architectural or system models. Its primary operational mode is trace-driven, enabling replay of real workload traces to evaluate memory-bound operations and kernel execution patterns within emerging CXL-attached architectures.

By combining accurate timing analysis with flexible, general-purpose NDP modeling, M^2NDP provides researchers with a practical and extensible platform for investigating the computational potential of memory-centric systems. Its modularity and adherence to CXL standards make it a valuable foundation for exploring next-generation heterogeneous systems, where memory devices play an increasingly active role in computation.

3.19. *PIMSIM-NN*

PIMSIM-NN [31] is a cycle-accurate and ISA-based simulation framework designed for evaluating PIM accelerators that execute deep neural network workloads. It addresses a key limitation of existing frameworks, where software and hardware models are tightly coupled, making it difficult to independently optimize compilation strategies or explore diverse hardware designs.

The central contribution of *PIMSIM-NN* is the introduction of a dedicated Instruction Set Architecture (ISA) that decouples software from hardware through a clean, abstract interface. Neural networks, described in ONNX format [32], are first compiled into ISA instructions by the accompanying *PIMCOMP-NN* compiler, which performs software-level optimizations and layer-to-hardware mapping. These ISA instructions are then executed by a cycle-accurate, event-driven simulator built on SystemC, which models configurable hardware parameters, realistic communication latency, and hardware parallelism through a reorder buffer mechanism.

This architecture enables flexible software–hardware co-design: researchers can test different compiler mappings without modifying the simulator, or sweep hardware configurations to assess energy, latency, and throughput trade-offs. Experimental validation shows that *PIMSIM-NN* captures realistic communication overheads (40–90%), in contrast to prior simulators that underestimate this effect. By providing modularity, configurability, and

open-source accessibility, PIMSIM-NN establishes a practical and extensible platform for the quantitative evaluation of PIM-based neural network accelerators.

3.20. *AiM Simulator*

AiM Simulator [33] is a performance and power simulation framework specifically developed for Processing-in-Memory (PIM) and Processing-Near-Memory (PNM) architectures targeting large-scale artificial intelligence (AI) inference workloads, particularly large language models (LLMs). It serves as the foundation for the CENT (CXL-ENabled GPU-Free sysTem) project [33], which aims to replace conventional GPU-centric computation with memory-centric processing paradigms. The simulator models a hierarchical system of Compute Express Link (CXL)-attached memory modules, where each module integrates near-bank PIM units optimized for dense linear operations (e.g., multiply–accumulate) and PNM units composed of lightweight RISC-V cores for non-linear functions such as normalization and Softmax. Through support for key CXL communication primitives (send, receive, and broadcast), **AiM Simulator** captures both intra-module and inter-module data movement, enabling realistic evaluation of distributed inference across large-scale multi-stack memory systems.

Architecturally, **AiM Simulator** performs cycle-level modeling of DRAM timing, PIM/PNM computation, and CXL link delays, while maintaining scalability across multiple modules. It also incorporates bandwidth and coherence constraints intrinsic to the CXL protocol, allowing researchers to investigate host–device communication overheads during neural layer offloading. Moreover, the simulator facilitates exploration of various parallelization strategies—pipeline, tensor, and hybrid parallelism—adopted in CENT-style systems. Integrated power and cost models allow users to estimate latency, throughput, and energy consumption under different hardware configurations, including link bandwidth, number of modules, and compute density.

Experimental evaluations from associated studies demonstrate that the CENT architecture, modeled using **AiM Simulator**, achieves more than $2\times$ higher inference throughput and approximately $3\times$ better energy efficiency than NVIDIA A100 GPUs for large-scale LLM inference, while also revealing critical performance bottlenecks imposed by CXL bandwidth and latency. By combining detailed timing models for PIM/PNM devices with accurate interconnect and power modeling, **AiM Simulator** provides a versatile and prac-

tical platform for designing, analyzing, and optimizing GPU-free, memory-centric architectures for next-generation large-scale AI systems.

3.21. 3D CiM LLM Sim

3D CiM LLM Sim [34] is a high-level, functional performance simulator designed to evaluate large language model (LLM) inference—including both dense and Mixture-of-Experts (MoE) architectures—on a conceptual 3D Analog In-Memory Computing (AIMC) accelerator. Its objective is to characterize latency, energy efficiency, and memory utilization when executing transformer-style networks on vertically stacked, high-density non-volatile memory substrates.

The simulator operates through a structured four-stage pipeline encompassing model definition and mapping, graph tracing, graph processing, and execution scheduling. Initially, the LLM model is defined in PyTorch and mapped onto analog compute tiles and digital processing units using a greedy utilization strategy that maximizes hardware efficiency. The system then leverages `torch.fx` to extract a fine-grained computational graph. During graph processing, high-level matrix–vector multiplication (MVM) operations are decomposed into tier-level sub-operations to accurately represent the 3D AIMC hierarchy. In the final scheduling phase, a custom instruction scheduler enforces architectural constraints such as one-tier-at-a-time (OTT) execution and bounded DPU or multi-head attention (MHA) availability, producing cycle- and resource-aware execution timelines.

While **3D CiM LLM Sim** models computational and dataflow behavior in detail, it intentionally abstracts interconnect contention and assumes a constant per-bit transfer cost between compute and memory tiers to preserve architecture generality. This design choice allows for flexible adaptation to multiple 3D AIMC configurations while avoiding overfitting to a specific interconnect implementation.

3D CiM LLM Sim offers a flexible, modular, and execution-driven simulation environment that enables early-stage architectural exploration of analog and hybrid 3D in-memory accelerators. It serves as a practical tool for analyzing LLM inference performance across different AIMC hierarchies, providing key insights into latency–energy trade-offs, memory access patterns, and utilization efficiency in next-generation compute-in-memory designs.

4. Simulation Perspectives: Functional vs. Timing Simulators

Understanding the distinction between functional and timing simulators is fundamental to accurate modeling and verification of modern digital and memory-centric systems. These two perspectives are inherently complementary: functional simulators ensure the correctness of logical behavior, whereas timing simulators validate performance and reliability under realistic microarchitectural constraints. As system complexity grows and timing margins shrink, leveraging both perspectives becomes essential for robust and efficient PIM design and validation. The classification of PIM simulators into functional and timing-oriented categories, along with representative examples, is presented in Table 1, while the subsequent sections provide an in-depth discussion of each simulation perspective.

Table 1: Functional and Timing PIM Simulators

Simulator Perspectives	Representative Simulators
Functional	PIMeval, PIMSim (Fast), NeuroSim, MemTorch, UPMEM SDK functional Sim, 3D CiM LLM Sim, UPMEM LLM Framework, MNSIM 2.0
Time-based	NDPmulator, PIMSim (Full, Inst), M ² NDP, PIMSimulator, uPIMulator, MPU-Sim, SMCSim, NVmain, PIMSIM-NN, MultiPIM, Ramulator-PIM, PiMulator, AiMsimulator

4.1. Functional Simulators

Functional simulators model the logical behavior of an instruction set architecture (ISA) or computational model without considering the temporal evolution of events. Instructions are executed in program order, ensuring architectural correctness—inputs and outputs match those of the real hardware—but without modeling microarchitectural timing effects such as pipelining, cache latency, or DRAM access delay.

Since functional simulators omit timing behaviour, they are lightweight and fast, making them particularly useful in the early stages of PIM designs. Researchers rely on them to validate the correctness of a new ISA extension, test compiler or runtime modifications, and ensure that software stacks execute properly on top of PIM instructions [8, 9]. Virtual platforms and early MPSoC simulators are typical examples of this class of simulators.

The trade-off is that functional simulators cannot capture performance-critical phenomena such as cache contention, memory bank conflicts, or the overlap between host CPU execution and in-memory computation. They

also often limit themselves to user-level execution, skipping OS-level behavior such as interrupts or system calls [11].

Although functional simulators do not capture detailed cycle-level timing, many modern PIM-oriented frameworks integrate analytical delay and latency estimators to provide approximate performance insight without the complexity of full microarchitectural modeling. Rather than advancing time cycle by cycle, tools such as `NeuroSim`, `MemTorch`, `PIMeval` and the Fast Mode of `PIMSim` utilize closed-form analytical equations or calibrated look-up tables based on `SPICE` [35] or `CACTI` [36] data to estimate access delay, energy, and throughput for memory-array operations [37, 38, 39, 40].

At this abstraction level, latency is computed from analytical models that relates circuit and architectural parameters such as cell resistance, wordline and bitline capacitance, and data reuse ratio, rather than through detailed cycle-by-cycle timing updates.

This hybrid analytical approach effectively bridges the gap between purely functional validation and comprehensive timing simulation, enabling functional tools to maintain high simulation speed while delivering quantitatively meaningful performance trends for efficient design-space exploration.

In summary, functional simulators are primarily designed to verify the correctness of system behavior, providing a fast and efficient means to validate functional accuracy. However, they lack the detailed timing information necessary to accurately assess performance metrics and provide the critical insights required for comprehensive performance analysis and architectural optimization. In contrast, timing simulators incorporate cycle-level timing details, enabling precise evaluation of performance characteristics such as latency, throughput, and energy efficiency. This distinction makes timing simulators essential for quantifying how much performance PIM architectures can deliver relative to traditional CPU and GPU baselines.

4.2. Timing Simulators

Timing simulators extend beyond functionality by modeling how instructions progress through microarchitectural structures such as pipelines, memory hierarchies, interconnects, and PIM units. Often called performance simulators, they provide detailed metrics such as latency, throughput, and energy efficiency. In this era, two main approaches exist: a) cycle-level and b) event-driven simulations, which will be discussed in detail below.

4.2.1. Cycle-level simulation

This kind of simulation advances the system one cycle at a time, updating every modeled component. It offers high fidelity by capturing pipeline hazards, memory scheduling, and contention making it particularly useful for studying fine-grained CPU–PIM interactions. However, cycle-level models are computationally expensive and memory-intensive, limiting their practical use to shorter workloads or micro-benchmarks [10, 9].

Cycle-based timing simulators usually achieve cycle accuracy by employing event scheduling and timing progression governed by validated memory timing backends that model the detailed behavior of modern DRAM and NVM subsystems. In practice, many PIM and near-memory simulation frameworks rely on well-established backends such as **Ramulator** [41], **DRAMSim2/3** [42, 43], **HMCSim** [44], **NVMain** [1] and **BookSim** [45] to maintain cycle-accurate synchronization across compute, interconnect, and memory domains. Each backend incorporates a dedicated timing engine:

- **Ramulator** models DRAM commands (ACT, PRE, RD, WR) and enforces JEDEC timing parameters at sub-bank granularity, enabling microsecond-scale trace replay with deterministic per-cycle scheduling.
- **DRAMSim2** introduced a transaction-level model where every DRAM command updates internal counters each clock cycle. **DRAMSim3** extends this to multi-channel, multi-rank DDR4/LPDDR4 systems with cycle-accurate queue arbitration and detailed power accounting.
- **HMCSim** abstracts 3D-stacked Hybrid Memory Cube interfaces, capturing vault controllers, link serialization, and TSV latencies at cycle granularity, enabling realistic timing analysis for near-bank PIM logic.
- **NVMain** targets non-volatile technologies (PCM, STT-RAM, ReRAM), combining cycle-accurate command scheduling with device-specific write/erase delays and endurance models.
- **BookSim** complements these by modeling network-on-chip (NoC) or interconnect fabric at a cycle level, synchronizing packet injection, router arbitration, and flit propagation with the memory backend.

As an example, **MultiPIM** and **Ramulator-PIM** leverage **Ramulator** to reproduce precise DRAM command scheduling and vault-level contention [7, 3]. **PiMulator** implements a DRAM controller RTL on FPGA, calibrates

its parameters against DRAMSim3 for clock-synchronous operation [5], while **MPU-Sim** adopts BookSim for network timing and couples it with DRAMSim2 for memory access latency modeling [23].

Conversely, some simulators achieve cycle accuracy without relying on existing backend. For instance, **uPIMulator** employs a fully custom event scheduler that models every DPU pipeline stage and DDR transaction explicitly in its timing engine, reaching hardware-validated precision through direct comparison with real UPMEM modules [27].

Such cycle-accurate architectures are essential for evaluating fine-grained interactions between compute pipelines, memory controllers, and in-memory execution units, providing deterministic timing behavior essential for hardware-software co-designs verification.

Hybrid approaches often combine cycle-level memory modeling with functional representations of computational circuits to balance accuracy and simulation efficiency [10]. In this work, we adopt cycle-accurate models consistent with these hybrid methodologies.

4.2.2. Event-driven simulation

This kind of simulation models advance simulation time by skipping idle cycles and progressing only on relevant events, e.g., memory requests or PIM kernel calls. This significantly improves simulation speed and scalability, albeit with reduced cycle-level detail making it particularly well-suited for large workloads and application-scale PIM evaluations [8].

Unlike cycle-accurate simulators that update system state every clock cycle, event-driven simulators manage discrete events—such as command completions, kernel activations, or DMA transfers—and focus computational resources solely on operations that impact architectural state. This paradigm has gained prominence in large-scale and heterogeneous PIM simulators, where full cycle-level modeling becomes computationally prohibitive.

Among contemporary PIM simulators, only a subset implements true event-driven kernels. For example, **LLMServingSim** [46] explicitly builds upon the **ASTRA-Sim** 2.0 [47] infrastructure, which utilizes a discrete-event engine to coordinate interactions among GPUs, NPUs, and PIM with timing granularity ranging from milliseconds to nanoseconds. Simulation time in this framework advances through event queues such as “kernel start,” “DMA completion,” and “token generation finish” rather than cycle-by-cycle, rendering it event-driven with cycle-level granularity but not strictly cycle-accurate.

LLMServingSim extends its event-driven co-simulation to heterogeneous

architectures that integrate PIM modules for accelerating memory-bound GEMV operations in LLM attention layers. This feature enables evaluating NPU–PIM cooperative execution within large-scale LLM serving environments [46].

PIMulator-NN [48] introduces its own event queue core (EDSC), where every computation and data-transfer module generates “START,” “STOP” and “FINISH” events. Latency and energy consumptions for these events are estimated using backend analytical models (e.g. **NeuroSim**, **CACTI**, and **NVSim**). This makes PIMulator-NN a canonical event-driven and cross-level simulator that integrates circuit-, architecture-, and algorithm-level timing without continuous clock stepping. However, since the source codes of these simulators are not publicly available, they are excluded from detailed examination in the subsequent chapters.

In the context of PIM research, timing simulators are indispensable for quantifying performance benefits such as the number of cycles saved by near-memory execution and the impact on bandwidth and energy efficiency. Consequently, event-driven and timing-accurate models are critical for rigorous performance evaluation of PIM architectures.

5. Simulation Scope: Full-System vs. Modular Simulation

Simulators can also be classified according to their scope and target, distinguishing those designed to model the entire hardware/software stack (full-system simulators) from those focusing on specific subsystems or user-level applications (modular simulators), as shown in Table 2. This distinction critically influences their applicability in PIM research, determining whether a tool is suited for comprehensive system-level evaluation or for isolated studies of memory-compute interactions.

Table 2: Full-System and Modular PIM Simulators

Simulator Scope	Representative Simulators
Full-System	PIMSim, NDPmulator, SMCSim, uPIMulator, MultiPIM, M ² NDP
Modular	PIMSim, PIMeval, PIMSimulator, MPU-Sim, PiMulator, AiM Simulator, PIMSIM-NN, NeuroSim, MemTorch, NVMain, UPMEM SDK functional Sim, UPMEM LLM Framework, 3D CiM LLM Sim, MNSIM 2.0, Ramulator-PIM

5.1. Full-System Simulators

Full-system simulators emulate an entire computing platform, encompassing processors, memory subsystems, I/O devices, and the operating system.

Such simulators are capable of booting an operating system within the simulation environment, allowing the execution of a complete software stack from kernel services to user applications, thus replicating real hardware behavior [9, 11].

This comprehensive modeling enables the investigation of OS-level effects—including system calls, virtual memory management, scheduling policies, and device drivers—which are particularly relevant for PIM architectures where near-memory computation integrates tightly with system software. A prominent example is gem5, which supports both full-system and syscall emulation modes.

In full-system mode, gem5 provides detailed analysis of PIM performance under realistic OS scheduling and memory management, whereas syscall emulation offers a lighter-weight environment for user-level studies [9]. Similarly, SimOS delivers a functional full-system simulation environment [49].

The principal limitation of full-system simulation lies in its complexity and computational overhead. Booting and managing an OS with all associated devices significantly increases simulation run-time, often confining full-system simulation to detailed case studies rather than rapid prototyping [10]. Nevertheless, when accuracy and realism are essential, such as for evaluating PIM accelerators integration with heterogeneous systems, full-system simulation is indispensable.

5.2. Modular Simulators

In contrast, modular simulators restrict their focus to specific modules or user-level applications, typically bypassing the operating system by intercepting system calls on the host machine [9, 8]. This narrower scope substantially reduces simulation complexity and accelerates execution.

Modular simulators are particularly valuable for targeted PIM investigations centered on memory subsystem behavior or in-memory computation, without the need to simulate full OS interactions. For instance, ZSim is a fast timing simulator for the x86 ISA that operates at the application level. Several PIM-oriented frameworks, including MultiPIM [7] and NATSA [50], extend ZSim with Ramulator [3] to model near-memory computation, focusing on memory subsystem dynamics and PIM kernel execution.

The trade-off for modular simulators is reduced accuracy, as they omit OS-level effects such as scheduling overheads, memory-mapped I/O, and interrupt handling, which can significantly impact server and data-intensive

workloads [10]. As a result, modular simulation is well-suited for rapid design space exploration of PIM primitives or microarchitectural components but inappropriate for assessing full-system deployment scenarios.

5.3. Case Study: PIMSim (*full-system and modular in one framework*)

PIMSim exemplifies a flexible simulator that integrates detailed PIM logic, memory devices, and coherence into a unified architecture. It combines DRAMSim2 [42], HMCSim [44], and NVMMain [51] backends, incorporates PIM-enabled instructions and code annotations, and supports both fine-grained (MESI) and coarse-grained coherence models. PIMSim provides three operational modes, each balancing fidelity and speed:

- Full-System mode: Executes user and kernel code by booting an OS inside gem5, modeling processors, memory, peripherals, and PIM logic in detail. It implements PIM pseudo-instructions and executes detailed PIM-enabled operations with coherence protocols (MESI, page, or LazyPIM-like[52]). This mode is essential when OS effects such as virtual memory, scheduling, and drivers critically influence PIM behavior.
- Instrumentation-driven (Pin-based) mode: Rather than simulating a full OS, PIMSim employs Intel Pin to monitor a running program on a host machine. It records dynamic instruction and memory behaviors as they happen and feeds them directly into the simulator, capturing realistic runtime dynamics with reduced overhead compared to full-system simulation.
- Fast (trace-driven) mode: This mode bypasses program execution entirely, instead replaying pre-recorded trace files containing logged memory accesses and instructions. This approach enables rapid evaluation of multiple hardware configurations but sacrifices the ability to model dynamic runtime effects.

6. Workload Feeding: Execution-driven vs. Trace-driven

Simulators can be categorized based on their input source distinguishing those that execute binaries directly (execution-driven) from those that replay pre-recorded instruction and memory traces (trace-driven). This classification has a significantly impact on simulator speed, accuracy, and flexibility,

Table 3: Execution- and Trace-Driven PIM Simulator

Workload Feeding Type	Representative Simulators
Execution-Driven	PIMeval, PIMSim (Full, Inst), NDPmulator, MNSIM 2.0, M ² NDP, PIMSimulator, uPIMulator, MPU-Sim, MemTorch, SMCSim, PIMSIM-NN, MultiPIM, NeuroSim, UPMEM SDK functional Sim, UPMEM LLM Framework
Trace-Driven	PiMulator, PIMSim (Fast), AiM Simulator, NVMain, 3D CiM LLM Sim, Ramulator-PIM

particularly in the context of PIM systems, where workloads often involve large-scale memory operations.

6.1. Execution-driven Simulators

Execution-driven simulators dynamically run benchmark binaries on the simulated architecture, integrating functional correctness with timing models. By directly executing programs, these simulators avoid reliance on pre-generated traces, simultaneously tracking performance metrics such as instruction latencies, memory accesses, and PIM kernel execution times [10][11]. This approach offers several advantages. It eliminates the overhead of generating and storing massive trace files, which is particularly beneficial for data-intensive PIM operations [9]. Moreover, execution-driven simulators can naturally capture behaviors dependent on runtime conditions, including speculative execution, dynamic scheduling, and branch mispredictions, which static trace-driven models cannot fully represent.

However, the complexity of maintaining a combined functional and timing engine makes execution-driven simulators typically slower and more challenging to develop [8]. Despite this, they are invaluable for analyzing runtime interactions between memory-side PIM operations and host processors, as well as for studying fine-grained effects such as coherence, consistency, and synchronization.

Several execution-driven frameworks have been proposed within the PIM domain. For instance, PiMulator executes workloads directly on a cycle-accurate FPGA-based SoC and models PIM timing alongside host processor interactions, enabling comprehensive studies of system-level phenomena such as coherence and runtime scheduling [5]. Similarly, PIMSim offers a full-system execution-driven mode that supports both user- and kernel- level code to simulate complete PIM-aware systems [2].

The UPMEM LLM framework integrates with PyTorch to overrides layer execution at runtime, directly executing models such as LLaMA or Mixtral,

while tracking compute, bandwidth and energy on a per-layer basis [53]. Likewise, M²NDP runs real RISC-V kernels and full PIM workloads cycle by cycle, modeling both functional correctness and detailed timing in CXL-attached near-data processors [30]. These tools demonstrate how execution-driven simulations enables observation of dynamic runtime effects, including synchronization, contention, and data orchestration between host and memory-side compute units, which are not accessible through static trace replay.

6.2. Trace-driven Simulators

Trace-driven simulators rely on pre-recorded instruction or memory-access traces generated from benchmark runs on real hardware or functional simulators [9]. These traces, comprising sequences of instructions, data addresses, and control flow outcomes, are input into a timing model to estimate performance (See Figure 3). This separation of functional and timing simulation offers a key advantage: once generated, traces can be reused to evaluate numerous architectural configurations without re-executing the program [8]. This reusability accelerates design-space exploration, which is particularly advantageous in PIM research where a single memory-access trace can be replayed across multiple timing models to compare near-memory computation schemes.

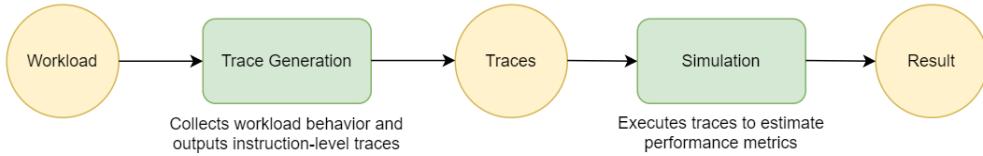


Figure 3: Abstract view of trace-driven simulation.

Nevertheless, trace-driven simulators encounter several limitations. Trace files can be extremely large and require significant storage and processing resources, sometimes necessitating sampling or compression techniques [10]. More critically, trace-based methods cannot runtime-dependent behaviors such as speculative execution or dynamic resource contention [11]. Consequently, while trace-driven approaches effectively characterize bulk memory access patterns and support comparative evaluations of PIM array designs, they may overlook crucial timing interactions between PIM modules and host CPUs.

Trace-driven methods are prevalent in PIM simulation due to their scalability. Tools such as **PIMSim** (fast mode), **Ramulator-PIM**, **NVMain**, and **AiMSimulator** rely on pre-collected traces to decouple functional and timing simulations. This enables rapid iteration over different memory organizations, near-memory instruction sets, and device-level latency models without costly benchmark re-execution [2, 3, 1, 33]. Additionally, trace-driven simulators are well-suited to large-scale AI workloads; for example, **AiMSimulator** and IBM’s 3D CiM LLM simulator generate instruction and memory traces from transformer-based LLMs, replaying these to evaluate PIM hardware performance in end-to-end inference tasks [34]. Similarly, **NVMain** supports non-volatile memory research by feeding in CPU-generated traces from simulators like **gem5** [1]. These examples highlight how trace-driven PIM simulators efficiently capture high-level dataflow patterns and energy-latency trade-offs, albeit at the expense of some fidelity in modeling runtime-dependent behaviors.

7. Abstraction Levels in PIM Simulation

Simulation of PIM systems encompasses multiple abstraction levels, each providing distinct insights and trade-offs between accuracy, complexity, and runtime. Understanding these abstraction layers is essential for selecting the appropriate simulation methodology depending on the research goals—ranging from high-level algorithmic performance analysis to detailed circuit- and device-level evaluations. We classify PIM simulation abstraction levels into five primary categories: Application level, System/Algorithm level, Architecture level, Circuit level, and Device level. Each of these abstraction levels is discussed below, and representative simulators are presented in Table 4.

7.1. Application Level

At the highest abstraction, application-level simulation models the behavior of end-user programs or workloads running on PIM-enabled systems. This level focuses on capturing application characteristics such as data flow, memory access patterns, and computational kernels without detailed consideration of hardware timing or architectural constraints. Application level evaluates end-user workloads such as neural networks, graph analytics, or Boolean processing.

In another word, this level’s simulators typically integrate with machine learning frameworks or runtime environments to analyze how PIM affects

Table 4: Abstraction Levels in PIM Simulation

Abstraction Level	Representative Simulators
Application	AiM Simulator, MNSIM 2.0, NeuroSim, MemTorch, Ramulator-PIM, UPMEM LLM Framework, 3D CiM LLM Sim
System / Algorithm	AiM Simulator, MNSIM 2.0, MultiPIM, NeuroSim, PiMululator, Ramulator-PIM, MemTorch, MPU-Sim, M ² NDP, NDPmulator, PIMeval, PIMSim, SMC-Sim, uPIMulator, UPMEM SDK functional Sim, PIMSimulator, UPMEM LLM Framework, 3D CiM LLM Sim
Architecture	AiM Simulator, MNSIM 2.0, MultiPIM, NeuroSim, NVMain, PiMululator, MemTorch, MPU-Sim, M ² NDP, NDPmulator, PIMeval, PIMSim, SMC-Sim, uPIMulator, UPMEM SDK functional Sim, PIMSimulator, 3D CiM LLM Sim
Circuit	AiM Simulator, MNSIM 2.0, NeuroSim, NVMain, PiMululator, MemTorch, MPU-Sim, M ² NDP
Device	MNSIM 2.0, NeuroSim, NVMain, MemTorch

overall application performance and energy consumption. It also provides application-level metrics (throughput, energy savings) on top of predefined PIM architectures (e.g., MNSIM, NeuroSim) [12]. This abstraction is particularly useful for early-stage exploration of workload suitability for PIM acceleration and for guiding architectural design choices based on observed algorithmic behavior [54].

7.2. System/Algorithm Level

System or algorithm-level simulation abstracts the PIM architecture to evaluate the impact of PIM operations within the context of the broader system or algorithm. This level models key components such as host processors, memory subsystems, and interconnects while incorporating simplified PIM kernels or instruction sets. System-level simulators capture interactions between PIM modules and the CPU, including data movement, scheduling, and resource contention, enabling the study of system-wide effects such as bandwidth utilization and latency reduction. [55]. At this level of abstraction, simulators demonstrate how operations or kernels (e.g., DNN layers, PIM instructions) are mapped and scheduled onto PIM hardware (e.g., PIMSim, PIMulator-NN) [12]. This level balances simulation speed with the ability to analyze performance and scalability of various algorithms when offloaded to memory-side compute units.

7.3. Architecture Level

Architecture-level simulation provides detailed modeling of the PIM microarchitecture, including components such as memory controllers, compute

units, interconnect fabrics, and coherence protocols. This abstraction emphasizes cycle-level timing, pipeline behavior, and memory hierarchy interactions. It describes how PEs are organized into banks, channels, and interconnects. Explores system-wide design trade-offs such as scalability, bandwidth, and hierarchy, e.g., NVSim [12]. Architectural simulators enable precise evaluation of design trade-offs such as instruction set extensions, cache coherence policies, and near-memory processing logic [56]. They are essential for verifying the functional correctness and performance of proposed PIM designs and for guiding hardware implementation decisions.

7.4. Circuit Level

Circuit-level simulation models PIM at the granularity of logic gates, timing delays, and electrical characteristics of integrated circuits. This level involves detailed analysis of critical circuit blocks such as sense amplifiers, arithmetic logic units, and memory cell arrays, accounting for signal propagation delays, switching activities, and power consumption. Circuit simulators often use tools like SPICE to validate the timing and energy efficiency of proposed PIM circuits [57]. This abstraction is crucial for optimizing circuit designs, minimizing latency and energy, and ensuring reliable operation under process variation and environmental conditions [12].

7.5. Device Level

At the lowest abstraction, device-level simulation focuses on the physical and material properties of the memory technologies used in PIM, such as DRAM, SRAM, PCM, ReRAM, or STT-RAM [58]. This level models phenomena including charge storage, retention, resistance switching, and endurance under electrical stress [12]. Device simulators incorporate physics-based models to predict device behavior, variability, and failure modes, which directly influence circuit and architectural design choices. Accurate device-level models are indispensable for emerging memory technologies and for enabling reliable integration of novel memory devices in PIM architectures [59].

This hierarchical framework highlights the complementary nature of abstraction levels in PIM simulation, enabling researchers to choose appropriate tools and methodologies based on their specific analysis requirements and design goals.

8. Memory Technologies in PIM Simulation

PIM architectures leverage a diverse range of memory technologies, each offering distinct features that must be carefully considered during simulation. Unlike high-level architectural models or workload abstractions, the choice of memory technology fundamentally constrains the supported operations, achievable performance, and energy efficiency of a PIM system. Consequently, memory should be treated as an independent and explicit simulation dimension rather than abstracted away within generic timing or workload assumptions [60]. In this section, we highlight memory technology as a central axis of PIM simulation, emphasizing that properties of the underlying storage medium are as critical as the algorithms executed upon it. Figure 4 and Table 5 respectively illustrate the memory technologies and the representative simulators evaluated in this study.

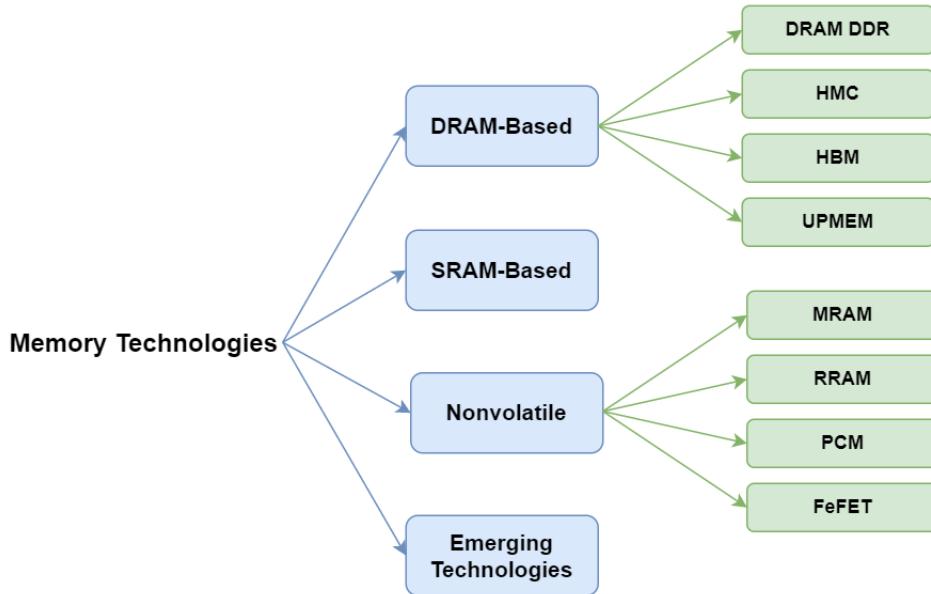


Figure 4: Memory Technologies in PIM Simulation.

8.1. DRAM-based PIM

DRAM-based processing-in-memory architectures represent a prominent class of near-memory computing solutions, exploiting the widespread availability and maturity of DRAM technologies. These architectures integrate

Table 5: Memory Technologies in PIM Simulation

Memory Technology	Representative Simulators
DRAM (DDR)	NVMain, MPU-Sim, PIMSim, Ramulator-PIM, PiMulator, PIMeval, NDPmulator
HBM	PIMSimulator, Ramulator-PIM, PiMulator, PIMeval
HMC	SMCSim (SMC), MultiPIM, PIMSim, Ramulator-PIM, NDPmulator
UPMEM	UPMEM SDK functional Sim, uPIMulator, UPMEM LLM Framework
SRAM-based	MNSIM 2.0, NeuroSim, NDPmulator
Non-Volatile Memories (NVM)	NVMain, NVSim, PIMSim, PIMSIM-NN, NeuroSim, MNSIM 2.0, 3D CiM LLM Sim, MemTorch
Emerging Technologies	M ² NDP (CXL), AiM Simulator (CXL)

computational capabilities directly within or close to the DRAM arrays to reduce data movement and improve energy efficiency. DRAM-based PIM designs can be broadly categorized based on the underlying memory interface and organization. Traditional DDR-based PIM approaches extend conventional double data rate memory with lightweight processing units, while Hybrid Memory Cube (HMC) and High Bandwidth Memory (HBM) architectures adopt 3D-stacked DRAM with logic layers, enabling more sophisticated and tightly integrated PIM capabilities. Separately, UPMEM introduces a commercial PIM solution embedding general-purpose RISC-V cores within DRAM chips, offering a programmable and scalable platform for a range of memory-centric workloads. In the following subsections, we discuss the key characteristics, advantages, and challenges of DDR, HMC, HBM, and UPMEM-based PIM systems.

8.1.1. DRAM DDR

Modern DRAM is organized hierarchically, exemplified by DDR (Double Data Rate) DRAM, which is accessed through multiple memory channels operating concurrently. Each channel has its own address, data, and command buses. One or more dual in-line memory modules (DIMMs) can populate a channel; each DIMM contains one or more ranks, and a rank is a set of DRAM chips that each contributing a subset of bits for each data transfer. Inside each chip, capacity and concurrency arise from multiple banks (typically 16 or more per chip); the same bank position across the chips in a rank forms a logical bank targeted by the memory controller [6]. Banks are further partitioned into subarrays, each with its own local row decoder and sense amplifiers to reduce electrical loading and improve timing. Hierarchical organization of a DRAM is shown in Figure 5.

Despite this internal parallelism, the off-chip interface is comparatively

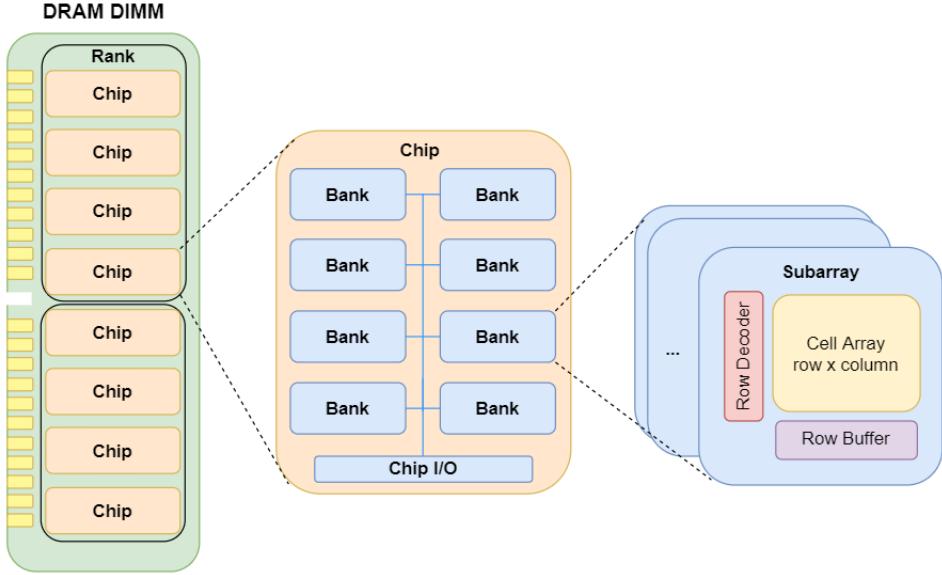


Figure 5: Hierarchical organization of a DRAM module.

narrow. For DDR the global data lines (GDL) at the bank interface are typically 64 to 128 bits wide, and pinout and signaling constraints on the channel limit the effective parallelism that can be exploited [6]. Modeling studies commonly assume practical module configurations, for example, an x8 rank with eight chips, to accurately map the memory structure to bandwidth and capacity parameters [60].

For scale, many DRAM models used in PIM research assume around 16 banks per chip and dozens of subarrays per bank, with subarrays sized in the hundreds to a few thousand rows and several thousand columns. These assumptions provide simulators a realistic range of structural parallelism for mapping PIM kernels without committing to a specific vendor's implementation [6].

Several well-known simulators model or emulate PIM behavior in DDR memories. NVMain, one of the earliest architectural-level simulators for emerging non-volatile and DRAM technologies, provides cycle-accurate modeling of timing, power, and hybrid DRAM–NVM configurations [1]. Although originally designed for PCM and MRAM, its flexible controller and timing model make it a common baseline for DDR-based PIM studies.

PIMSim builds directly on conventional DDR timing models (through

DRAMSim2) and integrates programmable PIM cores within DRAM banks, allowing detailed evaluation of PIM execution, coherence, and data movement under standard DDR constraints [2]. Similarly, **Ramulator-PIM**, an extension of the Ramulator memory simulator, adds in-memory compute units to DDR and other DRAM types [3]. It models the interactions between host processors and PIM cores through realistic DDR command traces, providing insight into bandwidth utilization and timing contention in 3D-stacked or multi-channel DDR environments.

Recent DDR-focused simulators have become increasingly detailed and heterogeneous. **PIMeval** introduces a unified performance and energy modeling framework for digital DRAM-PIM architectures, including subarray-level bit-serial, bit-parallel, and bank-level PIM designs within standard DDR4 configuration [6]. It provides a portable API and a benchmark suite that can evaluate different DDR-based architectures under uniform conditions.

8.1.2. HMC

The Hybrid Memory Cube (HMC) is a 3D-stacked DRAM architecture comprising multiple DRAM layers stacked on top of a dedicated logic layer, connected vertically via through-silicon vias (TSVs) and micro-bumps, as illustrated in Figure 6. The logic layer hosts the performance-critical control logic and partitions the stack into vertical “vaults,” each managed by an individual vault controller; a global controller coordinates vaults within the logic layer. Since timing is managed internally within the device, the host interface is simplified compared to conventional DRAM. HMC differs from High Bandwidth Memory (HBM) primarily in its host interface: HMC exposes packet-based serial links optimized for CPU-friendly control and programming, while HBM uses wide parallel links over a silicon interposer, commonly paired with GPUs [61].

This organization makes HMC an ideal substrate for near-memory processing. The logic layer fabricated using a logic-optimized process, can host compute blocks without the performance penalties typical of DRAM-compatible logic. Commonly, designs lightweight processing elements (PEs) are placed in the logic layer, one per vault, allowing each PE to access its local DRAM through short TSV paths. A mesh Network-on-Chip (NoC) facilitates inter-vault communication when necessary. This design achieves high internal bandwidth and reduces data-movement energy, though operations spanning distant vaults incur overhead due to remote traffic over the on-logic interconnect [61].

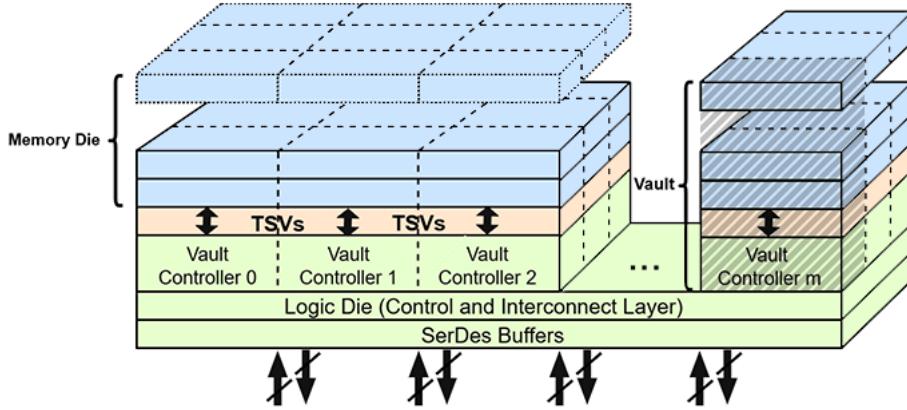


Figure 6: HMC architecture [62].

MultiPIM, a multi-stack PIM simulator models HMC-like stacked DRAM with vault controllers and internal networks. Each vault contains digital PIM cores executing memory-resident kernels. Computation is placed inside the vault logic layer, representing realistic near-bank processing within 3D-stacked DRAM cubes [7]. Through **PIMSim** configurable memory back-end, **PIMSim** can interface with **HMCsim** to emulate PIM in stacked DRAM [40]. Compute logic remains in-vault, digital, while the HMC model supplies multi-vault bandwidth and fine timing. The simulator supports comparing PIM across DDR and HMC generations.

In **Ramulator-PIM**, By switching Ramulator’s memory configuration to HMC, the same near-memory PIM cores operate over 3D-stacked DRAM timing. Processing occurs in the logic die of the cube or at vault controllers. It remains a digital model but benefits from HMC’s internal parallelism [3]. **MPU-Sim** framework simulates massively parallel near-bank processors (MPUs) mapped onto HMC-like or stacked DRAM systems [23]. Each vault hosts SIMD-style cores connected to the DRAM banks it manages. Compute thus resides near or within banks on the logic layer, exploiting stacked DRAM bandwidth.

The Smart Memory Cube (SMC) evolves the HMC concept by integrating programmable compute logic into the cube’s logic layer, transforming a passive memory controller into an active near-memory processing substrate [20]. Beyond vaults management, the logic die hosts lightweight process-

ing elements that perform arithmetic and data-reduction operations directly within the memory stack, minimizing data movement to the host. This enables fine-grained parallelism across vaults and exploits the cube’s high internal bandwidth for data-intensive workloads such as graph analytics and neural network inference. SMC thus serves as a practical intermediate step between conventional 3D-stacked DRAM and fully integrated PIM architectures, demonstrating how modest on-die compute capability can yield substantial energy efficiency and throughput gains [20].

SMCSim Designed for the Smart Memory Cube (SMC) platform, SMCSim reproduces full system behavior with compute elements embedded in the HMC logic die. Memory access obeys HMC vault timing, while compute instructions run in logic-layer digital cores. It bridges processor simulators and memory cubes to analyze near-memory workloads in stacked DRAM [20].

8.1.3. HBM

High-Bandwidth Memory (HBM) is a 3D-stacked DRAM technology integrated side-by-side with a host ASIC (CPU, GPU, or accelerator) in the same package via a silicon interposer. It stacks multiple DRAM layers atop a base logic die, connected through dense TSVs and micro-bumps and exposes an ultra-wide interface (typically around 1,024 I/Os per stack) that operates at moderate per-pin data rates (e.g., 3.2 Gb/s per pin for HBM2E). This combination of vertical stacking and wide, short interconnect yields very high device bandwidth with superior I/O energy efficiency compared to traditional PCB-attached DRAM. Current stacks commonly comprise 4 to 8 layers, with 12-layer stacks planned, and system designs typically deploy 4 to 6 HBM devices around a large SoC [63].

Samsung’s **PIMSimulator** models the organization and operation of HBM-PIM systems. The simulator reflects the internal hierarchy of an HBM stack composed of several memory banks and vaults connected to a base logic die that hosts lightweight Processing Units (PUs). These PUs execute matrix and vector operations near the memory banks, following the real HBM2/HBM3 timing protocols. **PIMSimulator** allows researchers to configure and analyze architectural parameters such as the number of stacks, PUs per channel, command scheduling policies, and memory bandwidth allocation. By doing so, it captures both the compute-in-memory behavior and the host-PIM coordination that define HBM-PIM execution [4].

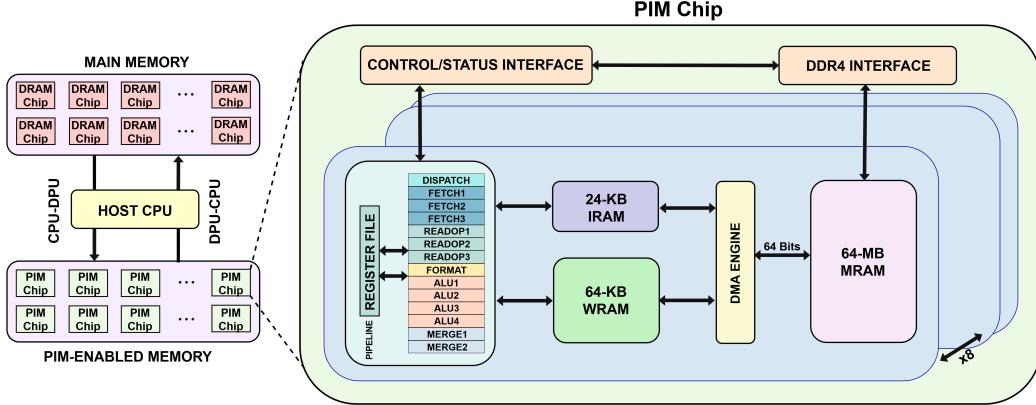


Figure 7: Overview of the UPMEM architecture, illustrating the interaction between the host CPU, main memory, and PIM-enabled memory [65].

8.1.4. UPMEM

UPMEM commercializes near-bank processing by integrating lightweight data processing units (DPUs) directly on the same die as commodity DDR4 DRAM (See Figure 7). A typical PIM chip combines 4 Gb DDR4-2400 DRAM with eight DPUs running at approximately 500 MHz, and DIMMs are built from 16 such chips. UPMEM reports about 2 TB/s achieves an aggregate internal DRAM-DPU bandwidth of about 2 TB/s at that scale, while retaining a standard DDR4 interface and largely unmodified DRAM process technology [64].

Each DPU is a simple scalar, in-order, multithreaded core designed to meet DRAM process constraints. To reach 500 MHz despite slow transistors, the microarchitecture uses a deep 14-stage pipeline and extensive fine-grained multithreading (24 hardware threads) to hide memory latency.

An autonomous DMA engine transfers code and data between DRAM (“MRAM”) and the on-chip SRAM. Programming follows a host-orchestrated model with a C/LLVM toolchain and runtime libraries; the CPU partitions data and dispatches kernels across thousands of DPUs [64].

uPIMulator is a cycle-accurate simulator of the commercial UPMEM DDR4-PIM architecture. Each DRAM chip integrates eight DPUs (14-stage in-order cores with IRAM/WRAM/MRAM) [27]. Compute happens inside each DPU, physically located within the DRAM chip banks. Memory technology is standard DDR4 and processing is digital and fully programmable. It reproduces microarchitectural details—register files, pipelines,

tasklets—validated against real hardware.

UPMEM SDK functional simulator, the functional simulator included in UPMEM’s official SDK emulates DPU behavior at the instruction level using real DDR4 timing profiles. It models in-DPU computation and standard DDR4 access delays but omits low-level electrical detail. The simulator is digital and serves as a fast functional tool for UPMEM programmers [21].

The **UPMEM LLM Framework** extends the official UPMEM SDK and simulator to support large-scale evaluation of Large Language Models (LLMs) on UPMEM’s PIM architecture. Built on UPMEM’s instruction-level functional simulator, it enables efficient mapping, scheduling, and distributed execution of neural network layers across thousands of DRAM Processing Units (DPUs) in multiple DIMMs. The framework faithfully reproduces functional behavior, memory hierarchy, and bandwidth constraints of the real hardware, allowing accurate estimation of performance trends and scalability. By integrating directly with the UPMEM compiler and runtime environment, it provides a practical platform for exploring memory-centric AI workloads and optimizing model partitioning and data movement before deployment on physical PIM systems [53].

8.2. SRAM-based PIM

Static RAM (SRAM) is the fast, on-chip workhorse of modern processor, storing bits using cross-coupled inverters (the classic 6T cell is shown in Figure 8). Unlike DRAM, SRAM does not require periodic refresh, enabling much lower access latency. However, SRAM cells are larger and more complex, trading density and cost for speed. As a result, SRAM is typically used in small, high-performance structures such as register files and CPU caches near the top of the memory hierarchy [61].

SRAM-based PIM focuses on in-cache and SRAM-macro computing, where operations are placed inside or next to cache/memory subarrays to exploit bit-level parallelism and low-latency access. **NDPmulator** can attach Near-Data Accelerators (NDAccs) at any cache level (e.g., L1/L2) as well as DRAM, enabling realistic, full-system studies of in-cache/SRAM PIM alongside the OS and host CPU [29]. Its scripts explicitly show NDAcc coupling to L2 and note that the same mechanism applies to L1 caches or DRAM, which is essential for modeling SRAM-resident PIM units within the cache hierarchy. **NDPmulator** also highlights prior in-cache processing work and contrasts it with its broader, multi-level support, reinforcing its suitability for SRAM-PIM evaluation across the hierarchy.

MNSIM 2.0 provides a unified array model that captures both analog and digital PIM. critically, it includes SRAM-based digital PIM and validates against a fabricated dynamic-logic SRAM PIM macro. The paper reports low error versus silicon ($\approx 3.8\%$ for SRAM-digital PIM) and details readout via sense amplifiers and bitwise logic (e.g., AND) attached to memory cells, precisely the mechanisms used in many SRAM PIM macros. This lets authors sweep array granularity, activated rows/columns, and interface resolution while staying faithful to SRAM-specific peripheral circuits and timing [26].

Complementing this, NeuroSim models SRAM-centric compute-in-memory pipelines and peripherals, including SRAM cell scaling trends, SRAM buffers, and energy/leakage accounting, features needed to compare SRAM-based digital PIM against eNVM options [24]. Its benchmark setup explicitly includes the ADC requirements for 1-bit SRAM cells, enabling apples-to-apples assessments of accuracy/efficiency across device choices.

Together, NDPmulator (system/OS level), MNSIM 2.0 (behavioral/architectural with SRAM-digital validation), and NeuroSim (device/circuit-to-system with SRAM support) form a coherent toolchain for SRAM in your taxonomy, spanning from cache-level integration to per-array circuit effects.

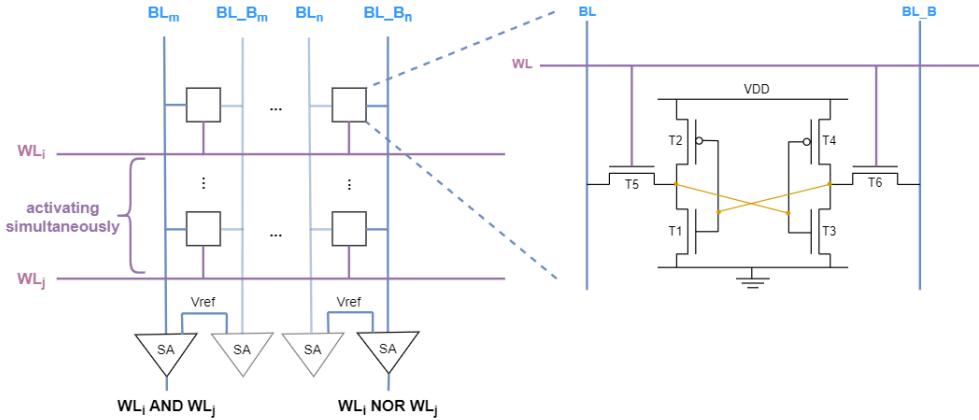


Figure 8: Bitwise logical operations in SRAM and a classic 6T SRAM cell [61].

These same properties make SRAM an attractive substrate for processing-in-memory. Digital in-SRAM (“bitline”) schemes repurpose thousands of cache arrays as fine-grain, bit-serial arithmetic logic units (ALUs), effectively turning last-level caches into wide vector engines. This approach reduces overhead from cache controllers and on-chip networks, enabling high

internal parallelism with one-cycle array accesses and minimizing data movement. Mixed-signal (analog) in-SRAM techniques go further by combining multi-row activation with lightweight DAC/ADC circuits, enabling vector-like operations directly on the shared bitlines. This can significantly boost energy efficiency and throughput when conversions costs are amortized [61].

8.3. Non-Volatile Memories

Non-volatile memories (NVMs) have emerged as promising technology for PIM architectures because they combine persistent data storage with the ability to perform computation directly within the memory array. Unlike DRAM, which requires refresh and destructive reads, many NVMs support non-destructive reads and can implement analog or digital logic in place. This reduces costly data movement and enables massive internal parallelism, offering energy-efficient computing beyond the traditional von Neumann model.

Some of prominent NVM technologies include magnetoresistive RAM (MRAM), resistive RAM (RRAM), phase-change memory (PCM), and ferroelectric FETs (FeFETs), each relying on distinct physical switching mechanisms. While these technologies offer advantages such as non-volatility, scalability, and compatibility with CMOS processes, they also face challenges such as endurance limits, variability, and write latency/energy trade-offs that must be addressed for practical large-scale systems [66, 67].

8.3.1. MRAM

Magnetic RAM (MRAM) cells are built around a magnetic tunnel junctions (MTJs), which consist of two ferromagnetic layers separated by a thin tunnel barrier. MTJ is a metal–insulator–metal (MIM) stack that consists of two ferromagnets separated by a thin tunnel barrier. One layer (“reference”) has a fixed magnetization, while the other (“free or storage”) layer is switchable [68]. The MTJ resistance depends on the relative relative orientation of these layers: parallel alignment yields a low-resistance state; anti parallel alignment yields a high-resistance state. Creating this relative orientation of the magnetizations of layers is known as tunneling magnetoresistance (TMR) effect. In practice, the free layer, tunnel barrier (often 1 nm MgO), and reference layer are co-designed to provide two well-separated, stable resistance states representing binary data [68, 69].

MRAM arrays typically use a compact 1T–1MTJ cell, where a select transistor gates read and write currents. MRAM combines non-volatility, high-speed read/write, and CMOS compatibility. However, designers must

manage bit-to-bit resistance variation and scaling challenges that tighten read margins as devices shrink [12].

8.3.2. RRAM

Resistor RAM (RRAM) stores information by modulating the resistance of a simple MIM stack, typically a thin metal-oxide layer sandwiched between electrode. An applied electric field creates and ruptures conductive filaments of oxygen vacancies within the oxide switching the device between a high-resistance state (HRS) and a low-resistance state (LRS) (SET and RESET operations, respectively) [70]. Filament dynamics are stochastic, causing SET/RESET thresholds to vary across write cycles and devices. Figure 9 a) depicts an RRAM cell, while Figure 9 b) demonstrates its use for in place vector–matrix multiplication.

RRAM is attractive for tightly integrated PIM because it can be fabricated in back-end-of-line (BEOL) interconnect layers with CMOS-compatible materials and stacked monolithically in 3D [70, 71]. While RRAM offers strong scaling potential and integration benefits, variability in switching thresholds and leakage in low-resistance states remain challenges, requiring adaptive write/read strategies and static power management [12].

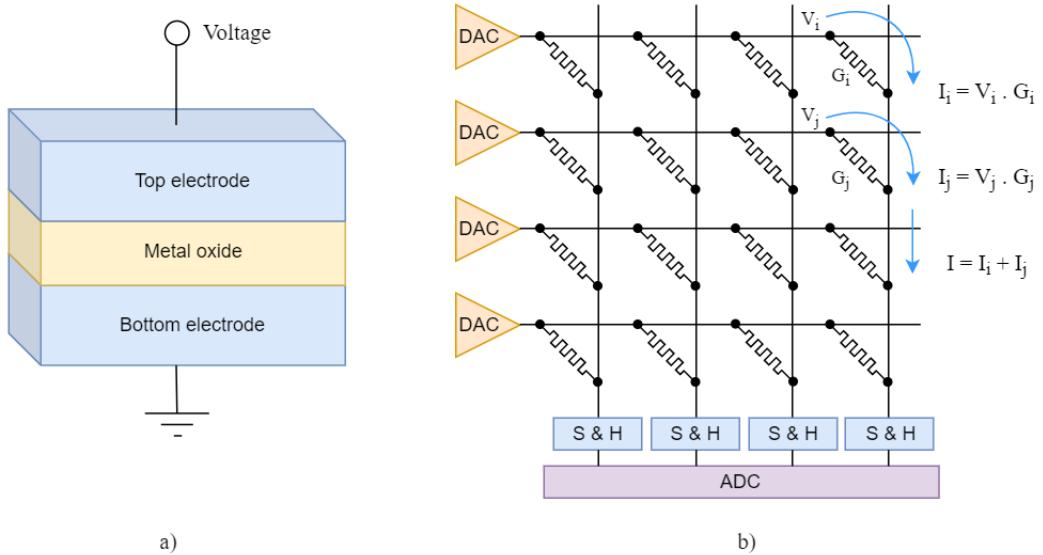


Figure 9: Nonvolatile memories: a) an RRAM cell, b) in place vector-matrix multiplication [58].

8.3.3. PCM

Phase-change memory (PCM) stores bits by switching a chalcogenide material between a high-resistance amorphous phase and a low-resistance crystalline phase, enabling significant read-current contrast and even multi-level storage [72]. Writing involves two thermal modes: SET recrystallizes the material by heating above the crystallization temperature, while RESET melts and rapidly quenches it to the amorphous state [72].

SET operations are typically slower nucleation and crystal growth, limiting write throughput. At the same time, PCM’s high operating temperatures and large RESET current complicate fabrication and scaling, as access devices must supply sufficient current interfaces must withstand thermal stress. Despite fast switching and strong read contrast, PCM designers must balance RESET power/speed with SET-limited write performance under tight thermal and process constraints [12].

8.3.4. FeFET

Ferroelectric FETs (FeFET) integrate a ferroelectric oxide layer into the MOSFET gate stack. Non-volatility arises from polarization hysteresis in the ferroelectric layer, with two stable threshold-voltage states encoding binary data [73]. FeFETs separate read and write paths, with write operations flipping ferroelectric polarization via gate-to-source voltage without current flowing through the drain–source channel. Voltage-driven writes result in lower operating energy compared to current-driven two-terminal NVMs such as RRAM, and FeFETs offer strong retention. However, endurance remains a significant concern for large-scale deployment [12].

NVSim provides parametric modeling for emerging non-volatile memories such as PCM, RRAM, MRAM, and FeFET. While it doesn’t execute compute, it quantifies latency, energy, and area of arrays that can later serve as analog or digital PIM blocks [19]. **NVMain** Complements NVSim with cycle-accurate timing for PCM, RRAM, and hybrid DRAM–NVM. It remains primarily a memory behavior simulator, but researchers have used it to attach PIM controllers to banks for near-array digital compute [1].

NeuroSim models analog in-memory computation using MRAM, RRAM, PCM, and FeFET crossbars. Computation is in-array analog multiply-accumulate (MVM) with peripheral ADC/DAC circuits. It spans device-to-architecture levels, capturing non-idealities like variation and quantization [24]. **MNSIM** 2.0 combining analog RRAM Computing In Memory and digital SRAM PIM

in one framework. Computation can occur analog inside crossbar arrays or near-array digital units depending on configuration [26].

MemTorch integrates PyTorch with models of RRAM memristive crossbars. It simulates in-array analog computation including stochastic and conductance-drift effects, allowing neural network training and inference analysis under realistic NVM behavior [25]. **PIMSIM-NN**, An ISA-based, cycle-accurate simulator for RRAM memristor-based PIM accelerators. It uses SystemC to orchestrate matrix (crossbar), vector, and scalar units. Computation occurs in-array analog MVM managed by digital control units. It supports ONNX models and models full chip timing [74]. **IBM 3D-CiM LLM Inference Simulator** for 3D analog in-memory computing, aimed at LLM inference. It abstracts multi-tier PCM/RRAM crossbars stacked vertically. Users supply latency/energy parameters for each analog operation. Processing is in-array analog, across multiple 3D layers, representing IBM's PCM-based AIMC research line [75].

8.4. Emerging Technologies

Compute Express Link (CXL) is not a memory technology, but a cache-coherent, high-bandwidth interconnect that enables memory devices and accelerators attach to the CPUs as coherent peers, as shown in Figure 10. For PIM, CXL facilitates packaging intelligent memory modules directly into the system fabric as first-class devices, enabling external memory-side processors that share coherent data structures with the CPU. This approach simplifies software integration compared to traditional DMA-based offload but introduces fabric latency ($\sim 100\text{--}200$ ns round-trip) and bandwidth constraints compared to native DDR access [30].

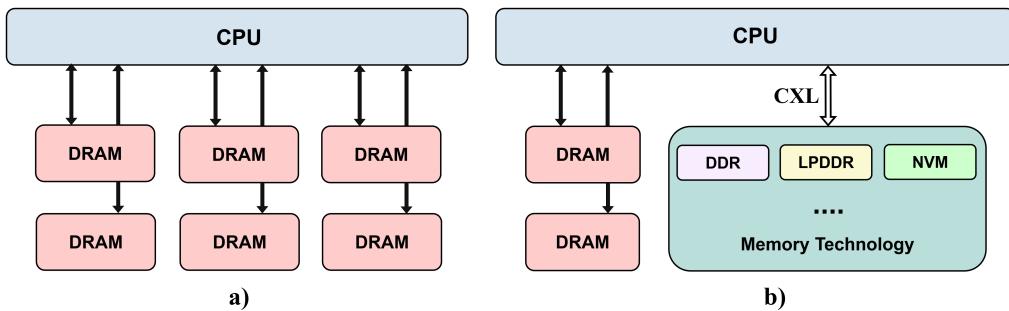


Figure 10: CXL attached memory [76]: a) without CXL, b) with CXL.

A forward-looking PIM simulator should model CXL link latency and bandwidth, protocol overheads including coherence traffic (such as cache invalidations/updates caused by PIM operations), and fabric topologies that enable distributed PIM modules cooperating over a switch. Programming models can treat such PIM modules as coherent peers or memory-mapped devices, reducing software complexity. Thermal/reliability concerns shift toward the interconnect components rather than memory cells, enabling disaggregated, pooled memory with integrated compute and expanding architectural possibilities for PIM.

M²NDP models Near-Data Processing inside CXL.memory expanders backed by DRAM (typically LPDDR5). Compute engines reside within the CXL device controller near the DRAM channels, using coherent CXL.mem protocols. Processing is digital and near-memory rather than in-cell [30].

AIM Simulator models a GDDR6-PIM system connected via CXL. Each DRAM bank embeds digital MAC arrays, while near-memory units in the CXL controller handle higher-level operations. Computation is split between in-bank PIM and near-memory digital cores. Memory technology is GDDR6 DRAM with modest logic integration [33].

9. Application Domains of PIM Simulators

PIM simulators have been evaluated across various application domains. Their classification based on application and usage largely hinges on the types of benchmarks employed in their validation studies or their overall use cases. Some simulators are highly domain-specific, concentrating workloads such as deep learning or graph analytics, while others are designed as general-purpose frameworks to accommodate a wide array of applications. In this section, we categorize PIM simulators into five primary groups: 1) AI and Machine Learning, 2) Graph Analytics, 3) Database and Data Analytics, 4) High-Performance Computing (HPC), and 5) Domain Agnostic or General Purpose. The visual overview of this classification is presented in Figure 11, along with Table 6, which illustrate representative PIM simulators.

9.1. AI and Machine learning

AI workloads, especially deep learning, are often memory-bound. Both training and inference involve massive matrix–vector multiplications (MVMs), convolutions, and tensor transformations. In GPUs and CPUs, moving data from DRAM to compute units dominates energy and latency: a single 64-bit

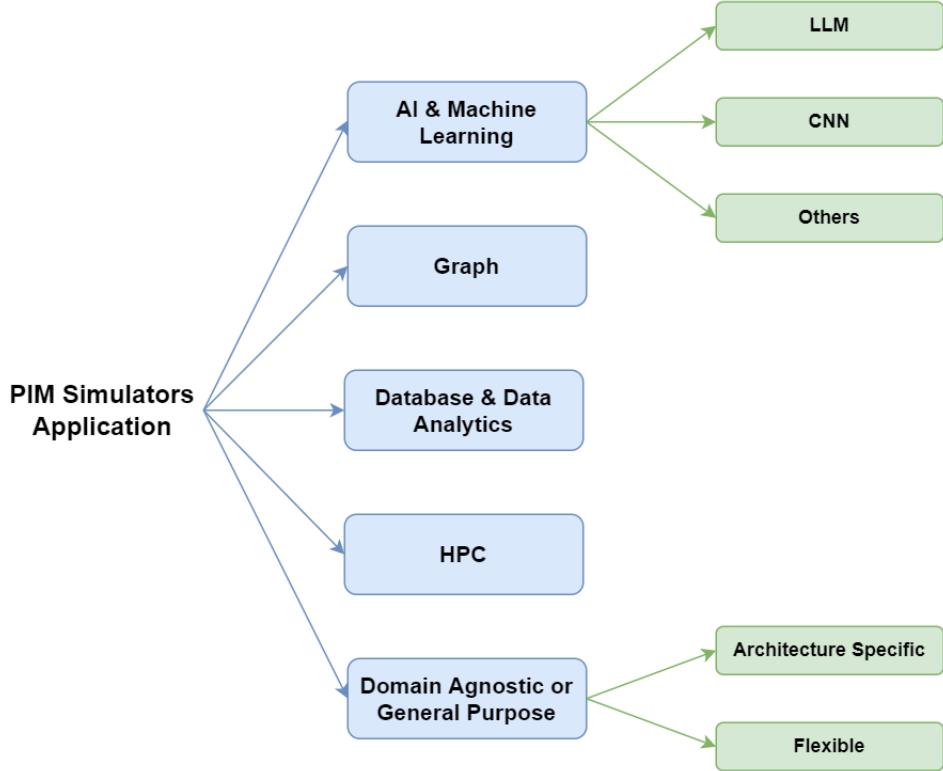


Figure 11: PIM Simulators Applications Classification

DRAM access can consume roughly 1000 times more energy than a floating-point multiply [15]. AI workloads impose distinct computational and memory access patterns during training and inference, each presenting unique challenges to system design. Moreover, the presence of both dense and sparse operations across various tensor formats necessitates flexible and efficient dataflow mechanisms [77]. The critical characteristics of these workloads can be summarized as follows:

- *Training*: Requires frequent weight updates, backpropagation, and gradient storage, which generate enormous read–write traffic between compute and memory [78].
- *Inference*: Models like CNNs, transformers, and large language models (LLMs) need large parameter fetches and key–value (KV) caches for attention, stressing bandwidth and memory capacity [79].

Table 6: Application Domains of PIM Simulators

Application Domain	Representative Simulators
AI / Machine Learning	PIMSim, PIMeval, MPU-Sim, PIMSim-NN, NeuroSim V2.1, MNSIM 2.0, uPIMulator, MemTorch, NDPmulator, M ² NDP, 3D CiM LLM Sim, PIMSimulator, AiM Simulator, UPMEM SDK functional Sim, UPMEM LLM Framework
Graph Analytics	SMCSim, PIMeval, PIMSim, Ramulator-PIM, MPU-Sim, MultiPIM, uPIMulator, M ² NDP
Database / Data Analytics	PIMeval, PIMSim, UPMEM SDK functional Sim, MPU-Sim, uPIMulator, NDPmulator, M ² NDP
High-Performance Computing	NVMain, SMCSim, PIMSim, PIMeval, Ramulator-PIM, MPU-Sim, PIMSimulator, uPIMulator, NDPmulator, M ² NDP, MultiPIM
General Purpose	uPIMulator, SMCSim, Ramulator-PIM, MPU-Sim, PIMeval, PIMSim, PiMulutor, NDPmulator, M ² NDP, PIMSimulator, MultiPIM, NVMain, UPMEM SDK functional Sim, UPMEM LLM Framework

- *Sparsity & Tensor Diversity:* Both dense (e.g., matrix multiplications in CNNs) and sparse (e.g., attention mechanism, recommendation systems) operations occur, demanding flexible dataflows [80].

Most simulators in this category focus on the accelerating General Matrix-Matrix Multiplication (e.g, PIMSIM-NN [31], AIHWKit [81]) and General Matrix-Vector Multiplication (e.g, PIMeval/PIMbench [39]). As a result, PIM helps because: a) avoids costly data transfers caused by heavy computations such as multiply-accumulate (MAC), pooling, and attention kernels; b) scales bandwidth with memory capacity; c) use emerging memories (ReRAM, PCM, FeFET) that natively support analog MVMs [82], well aligned with AI’s linear algebra backbone.

9.2. Graph

Graph processing algorithms involve irregular memory access, pointer chasing, and low arithmetic intensity. Operations such as neighbor traversal, frontier expansion, and vertex updates require fetching scattered data across large memory spaces [16]. Unlike AI workloads that are compute-intensive, graph workloads are often limited by random memory latency. Critical operations in Graphs include:

- *Breadth-First Search (BFS):* repeated queue/frontier expansion, heavy on atomic updates [83].
- *PageRank:* iterative floating-point updates to vertex ranks.

- *Shortest Path (Dijkstra/Bellman-Ford)*: frequent check-and-update operations on edges.
- *Graph Traversals (DFS, SSSP, Connected Components)*: pointer chasing across adjacency lists.

These operations cause bandwidth bottlenecks and high synchronization costs. CPUs and GPUs often stall on random DRAM accesses, while PIM architectures reduce traffic by performing updates (atomics, comparisons, rank increments) directly inside memory [84]. Currently, many PIM architectures are proposed to ease and speedup graph processing, such as Tesseract [16], GraphP [83], and GraphH [85]. Also, simulator like **HMC-Sim** [84], and **PIMeval** [39] support graph processing workloads.

9.3. Database and Data Analytics

Database management systems (DBMS) and analytics platforms are dominated by data movement rather than arithmetic computation. Typical queries such as joins, aggregations, scans, filters, require streaming or shuffling gigabytes of tuples across memory hierarchies. For example, TPC-H benchmarks show that selection, projection, join, and aggregation account for nearly 90% of total execution time and memory traffic[17].

On conventional CPUs and GPUs, data must travel from DRAM through caches into execution units, even if only simple comparisons or additions are performed. PIM mitigates this by executing common database primitives such as comparison, filtering, aggregation, and sorting, directly inside or near DRAM banks. This reduces latency, energy consumption and intermediate traffic. Critical Database Operations for PIM are as follows:

- *Selection/Filtering*: ideal for SIMD-style execution inside DRAM vaults.
- *Projection*: can be combined with filtering to reduce result size.
- *Aggregation and Grouping*: bandwidth-bound; atomic add/multiply inside memory reduces traffic.
- *Join*: extremely data-intensive; benefits from PIM’s parallel row scans and bulk bitwise operations.
- *Sorting*: bandwidth- and compute-intensive; PIM can accelerate sorting primitives.

To address the need for PIM databases, architectures such as **PIM-enabled instruction sets** [86] are proposed to show that common database operations like selection and aggregation can be executed within a 3D-stacked memory logic layer, avoiding the round trip to the CPU for simple comparisons and arithmetic. This study supports in-memory hash join, histogram and radix partitioning. In addition, **PIMeval/PIMbench** [39] specifically supports database and data analysis benchmarks.

9.4. High Performance Computing (HPC)

High-performance computing workloads are both memory- and compute-intensive requiring sustained high bandwidth, low latency access, and scalability across thousands of cores. Critical HPC operations include:

- Stencil Updates: iterative computations on structured grids [18].
- Sparse Matrix-Vector Multiplication (SpMV) [87].
- Fast Fourier Transform(FFT) and spectral transforms [88].
- Partial Differential Equation(PDE) solvers (wave propagation, elasticity).
- Linear Algebra Kernels (BLAS, LU/Cholesky factorization).

Traditional HPC nodes suffer from the memory wall, where interconnect and DRAM bandwidth cannot keep pace with core scaling. PIM simulators for HPC explore how HBM or HMC with in-memory compute can efficiently reduce traffic for these kernels.

Some of the research proposals in HPC include **Casper** [18], which accelerates stencil computations with specialized units placed near the CPU’s last-level cache to maximize local bandwidth. Other work like **TOP-PIM** [87] embeds programmable, throughput-oriented GPU cores within 3D-stacked memory to serve as PIM accelerators for kernels including SpMV. For bandwidth-bound tasks like FFTs, collaborative approaches such as **Pimacolaba** [88] are proposed, splitting the computation between the host GPU and PIM hardware to balance high bandwidth with compute power, thereby improving performance and reducing data movement. Simulators such as **PIMSimulator** [4], **HMC-Sim** [84], and **PIMeval/PIMbench** [39] support HPC-related workloads.

9.5. Domain Agnostic or General Purpose

These simulators evaluate a broad spectrum of workloads (AI, HPC, graph, database, analytics) rather than focusing on a single domain. Some are flexible frameworks (e.g., UniNDP [89]); others are architecture-specific (e.g., MIMDRAM [90], AttAcc [91], NeuPIMs [92]). They enable fair comparisons across PIM designs and system-level integration and exploration. Some are based on real architectures (e.g., UPMEM SDK [21], uPIMulator [93]), while others are abstract research tools providing configurable backends (e.g., PIMSim [2], MultiPIM [7]).

10. Contexts of PIM Simulator Deployment

PIM simulators today play diverse roles across academic research, industrial design, and education. While most tools originate within one of these domains, their applications often extend well beyond their initial scope. For example, academic simulators such as NeuroSim and NVSim [24, 19] have evolved into widely adopted industrial frameworks, while industrial simulators like UPMEM’s SDK are employed in university research and training purposes. Therefore, PIM simulators should not be regarded as exclusive to a single context; rather, they form a flexible ecosystem that adapts to diverse needs including architectural exploration, design validation, and education purposes. Broadly, three deployment contexts can be identified:

- Academic research focused on exploring new architectures and evaluating untested ideas.
- Industrial development aimed at product validation, pre-silicon optimization, and reliability assessment.
- Education and training dedicated to teaching, demonstration, and skills development.

Each context imposes specific requirements on simulation frameworks, including accuracy, scalability, usability, and integration with existing toolchains. The following sections discuss these contexts in detail and highlight representative simulators in each category.

10.1. Academic Research Applications

In academic research, PIM simulators serve as essential platforms for investigating architectures that bring computation closer to memory. They bridge the gap between theoretical design and hardware prototyping by providing controlled, low-cost, and reproducible environment for design-space exploration, performance analysis, and hardware-software co-design.

Modern academic simulators, such as `NDPmulator` and `MultiPIM`, offer system-level emulation of CPUs, memory hierarchies, and near-data accelerators under both user-space and kernel-space conditions [29, 7].

Cycle-accurate tools like `PIMSIM-NN` and `MNSIM 2.0` enable researchers to evaluate neural-network workloads with various quantization schemes, mapping strategies, and fault-tolerance mechanisms [74, 26].

A key recent trend is cross-layer simulation, where architectural frameworks integrate detailed circuit- or device- models. For example, `NeuroSim` and `NVSim` provide validated circuit- and device-level models that are often linked with architecture-level tools like `PIMulator-NN` or `MNSIM 2.0`. This integration enables end-to-end analysis of non-idealities, power consumption, and accuracy degradation in both digital and analog PIM designs [24, 19].

At higher abstraction levels, simulators such as `PIMSim` and `Ramulator-PIM` support flexible configuration of DRAM timing parameters, interconnect topologies, and offloading policies. These frameworks are widely used to evaluate PIM architectures [2, 3] and also CXL-based NDP architectures, as exemplified by `M2NDP` [30].

Thus, academic simulators provide a foundation for innovation by lowering experimental barriers. They enable research groups to evaluate emerging paradigms, such as 3D-stacked DRAM PIM, ReRAM-based analog computing, or hybrid CXL-PIM architectures, well before hardware fabrication, accelerating discovery in the field.

10.2. Industrial Applications

In industry, PIM simulators shift from exploration to verification, optimization, and integration within product design workflow. These tools emphasize cycle accuracy, determinism, and compatibility with commercial EDA or AI frameworks, serving as essential pre-silicon validation platforms.

A prominent examples is the `UPMEM SDK Functional Simulator`, part of the official development kit for UPMEM’s commercial DRAM-PIM chips [21]. It accurately models real DPU (Data Processing Unit) behavior, enabling developers to compile, debug, and profile applications using the same

APIs as on hardware. This simulator underpins both industrial and academic collaborations focused on programming models and workload characterization.

UPMEM’s **PIM-AI** framework extends this concept to Large Language Model (LLM) inference [28] integrating PyTorch-level workloads with realistic models of energy consumption, throughput, and total cost of ownership (TCO). This allows accurate performance estimation under cloud or mobile deployment scenarios. Similarly, **uPIMulator** offers cycle-level simulation of commercial UPMEM-PIM systems. Calibrated against hardware measurements, it has become a reference model for evaluating commercial DRAM-PIM technologies within the research community [27].

Samsung’s **PIMSimulator**, developed by the Samsung Advanced Institute of Technology (SAIT), provides a cycle-accurate model of HBM and logic-layer interactions [4], helping engineers in analyzing PIM instruction sets and memory behavior before silicon implementation. Industrial R&D centers have introduced simulators for specialized use cases as well. The **IBM 3D-CiM LLM Inference Simulator** models hybrid analog–digital compute-in-memory architectures with detailed 3D stacking and thermal considerations, integrated into **IBM’s AIHWKit** and chip design workflows [34, 81].

Several academic–industrial hybrid frameworks further blur the boundaries between research and product development. For instance, **PiMulator** is an FPGA-based emulation platform achieving up to 28 \times faster evaluation than software simulators, providing near-real-time prototyping [5]. Similarly, **M²NDP** (POSTECH & SK hynix) enables cycle-accurate modeling of CXL-based memory expanders with embedded computation [30].

At the device and circuit level, mature tools such as **NeuroSim** and **NVSim** remain cornerstones for both academia and industry, serving semiconductor vendors like Intel, SK hynix, and TSMC for technology evaluation [24, 19]. Overall, industrial PIM simulators focus on hardware prototyping, product validation, and design optimization. They provide deterministic, performance-calibrated models that shorten design cycles, reduce development cost, and mitigate fabrication risks.

10.3. Educational and Training Applications

Beyond research and product design, PIM simulators have emerged as valuable tools for education and professional training. They enable learners to observe the interaction of computation and data within memory hierarchies, connecting theoretical concepts to measurable system behavior.

Simplified frameworks such as DRAMSim3, Ramulator-PIM, and PIMSim are used in undergraduate courses to visualize memory operations like row activation, bank-level parallelism, and near-data execution [43, 3, 2]. By conducting simple experiments, students can directly observe how PIM reduces data movement and improves energy efficiency.

At the graduate and research level, tools like MemTorch, NeuroSim, and PIMSIM-NN facilitate hands-on exploration of neuromorphic and compute-in-memory accelerators. Their Python and PyTorch integration make them ideal for understanding co-design principles, circuit variations, and algorithmic resilience [25, 24, 74].

Industrial frameworks, including UPMEM’s SDK and IBM’s 3D-CiM simulator, are also valuable for corporate training, helping engineers gain familiarity with data-centric computing and heterogeneous memory systems.

These environments provide realistic experimentation without requiring physical access to hardware, making them ideal for skill development and technology adoption. Educational deployment of PIM simulators thus bridges theory and practice, preparing a new generation of architects and engineers capable of integration software insight with hardware innovation.

11. Evaluation Metrics for PIM Simulation

The rigorous evaluation of modern memory architectures, whether conventional DRAM, emerging NVM, or tightly coupled Processing-in-Memory accelerators requires a comprehensive set of metrics. These metrics range from foundational processor performance indicators such as Clock Per Instruction (CPI) to specialized characteristics related to power, area, and reliability specific to the underlying memory technology. Simulator frameworks integrate these metrics across multiple abstraction levels, from device physics to the application layer, to enable efficient design space exploration.

11.1. Performance Metrics

Performance metrics remain the cornerstone of architectural evaluation. They quantify how quickly and effectively a system executes workloads and processes data. These metrics encompass aspects such as latency, throughput, and bandwidth, each essential for characterizing overall system behavior and identifying the bottlenecks.

11.1.1. Core Latency and Execution Efficiency

This category of metrics measures the fundamental speed of the processing elements, ranging from host CPU to PIM cores. It also accounts for memory access latency and communication latency across interconnects.

Execution Time and IPC/CPI: The most fundamental metric used to assess performance is *execution time* (T), which is extracted as follows:

$$T = N \times CPI \times \frac{1}{f}$$

where N is the total number of executed instructions, CPI represents cycles per instruction, and f is the operating clock frequency [11].

Closely related metrics such as Instructions per Cycle (IPC) and its reciprocal, CPI, highlight utilization efficiency of the execution pipeline. CPI breakdowns are used to isolate delays due to stalls, cache misses, or pipeline hazards [11].

In PIM architectures, total latency is often decomposed into *data movement latency*: the time spent transferring data between the CPU and memory, and *PIM kernel execution latency*: the time spent executing computational operations directly within memory subsystems [39].

Speedup: Speedup is a universal metric used to express the performance improvement achieved by a baseline (e.g., CPU or GPU).

The theoretical limits of speedup are dictated by Amdahl's Law, which states that the improvement is bounded by the fraction of the program that can be parallelized. Therefore, only the parallelizable components of a workload, such as PIM-accelerated computation or optimized DRAM access, contribute to meaningful speedup [11].

Latency Stacks and Timing Breakdown: Simulators often employ latency stack diagrams to visualize the cumulative delays in workload execution. These stacks typically include, queuing, DRAM access, and compute latency. Such visualizations help in identifying dominant bottlenecks in the system and are critical for architectural tuning and optimization.

11.1.2. Memory Bandwidth and Throughput Analysis

These metrics are tailored to evaluate the performance of the memory subsystem, particularly in memory-bound applications where the memory wall becomes the limiting factor.

Bandwidth Utilization: Bandwidth utilization measures the ratio of achieved memory bandwidth to the theoretical peak bandwidth. Low utilization in a memory-intensive workloads highlights bottlenecks in the data path, interconnect, or memory controller scheduling.

Workload Parallelism and Scaling (STP/ANTT): In systems supporting multiprogramming or multithreading with shared DRAM access, metrics such as System Throughput (STP) and Average Normalized Turnaround Time (ANTT) are vital. STP quantifies the aggregate throughput across concurrent tasks, while ANTT reflects fairness and latency per workload.

These metrics are critical for characterizing DRAM access conflicts, the impact of non-determinism, revealing the trade-offs between maximizing overall bandwidth usage and ensuring fair resource allocation across concurrent tasks [11].

11.2. Power and Energy Metrics

As data movement becomes the dominant contributor to system-wide power consumption, energy metrics have grown in importance, often becoming the ultimate constraint for designing scalable and sustainable architectures. These metrics span from low-level circuit energy use to application-level energy footprints.

11.2.1. Energy Consumption and Modeling Methodology

Accurate power estimation requires integrating detailed power models into cycle-accurate simulators.

Dynamic and Leakage Power: Dynamic Power arises from transistor switching activity during logic operations and memory access. Leakage Power represents static loss during idle states. In NVM-based PIM, leakage power is typically much lower than DRAM, allowing aggressive power-gating techniques [19].

Energy Efficiency (TOPS/W): Tera Operations Per Second per Watt (TOPS) is used for measuring the efficiency of AI hardware accelerators. It allows fair comparison across systems with varying data types and operational scales by normalizing performance against power consumption [26].

11.2.2. Physical and Economic Constraints

Beyond energy, architectural feasibility also depends on physical layout and cost.

Area: Silicon area is measured in mm² and includes core logic, interconnects, and memory arrays. It often scales with the fabrication technology node (F). Tools like CACTI [36] provide joint estimates of area, timing, and leakage to support design space exploration with trade-offs between performance density and cost [19].

11.3. Thermal and Reliability Metrics

To ensure long-term operational correctness and safety, modern simulators must also model temperature dynamics and component-level reliability. These factors are especially crucial in 3D-stacked and high-density memory systems like PIM.

11.3.1. Reliability and Lifetime

Reliability encompasses the ability of memory elements and compute units to maintain correct operation over time. This metric is measured based on various parameters, as outlined below.

Endurance (Write Cycles): Endurance is measured by the maximum number of program/erase or write cycles a memory cell can undergo before it becomes unreliable. This is particularly important for NVM technologies like Phase-Change Memory (PCM) and Resistive RAM (ReRAM), where limited endurance can restrict applicability in high-write environments [19, 94].

Data Retention: This metric refers to the duration for which a memory cell can reliably hold data without refresh. While DRAM requires frequent refresh due to its volatile nature, NVM technologies offer significantly longer data retention, making them suitable for persistent memory applications [19].

Process Variation and Non-Ideal Factors: At the device level, simulators model the effects of *a*) manufacturing variability (e.g., Gaussian resistance noise or threshold voltage); *b*) permanent defects (stuck-at-faults). These non-idealities are particularly relevant in analog or low-precision digital computation, as they can introduce significant functional errors in PIM circuits [95].

11.3.2. Thermal Stability and Functional Accuracy

Thermal management ensures the physical stability of the chip, while accuracy verifies the correct functionality of the resulting computation.

Thermal Modeling: Thermal simulations track heat dissipation and peak operating temperatures across memory chips. In DRAM, elevated temperatures

require more frequent refresh operations, reducing overall performance and increasing energy use. Tools like DRAMSim3 [43] and HotSpot [96] are often used together to co-model access scheduling and thermal behavior.

Functional Accuracy: For AI accelerators, particularly those executing Convolutional Neural Networks (CNNs), functional accuracy becomes a critical metric. Since PIM hardware may use low-precision data types or analog computation (which introduces quantization errors and noise), simulators must include full training or inference workflows to assess the impact of hardware constraints on model accuracy.

12. Benchmarking Methodologies for PIM Architectures

In the realm of PIM architectures, benchmarks serve as a critical foundation for research and evaluation. They provide a standardized measurement framework that allows researchers to evaluate and compare different designs, algorithms, and simulators under standardized uniform conditions. Without such standardized benchmarks, studies would resort to disparate workloads, inconsistent metrics, and varying inputs, making cross-comparison and reproducibility nearly impossible. Benchmarks offer several critical benefits such as:

- *Reproducibility*, shared workload ensure experiments can be repeated and results independently validated.
- *Comparability*, by evaluating architectures on the same benchmark set, results gain semantic meaning, allowing for fair, side-by-side comparisons across papers or research efforts.
- *Stress Testing*, well-designed benchmarks expose architectural strengths and weaknesses, whether they are memory-bound, compute-bound, or suffer from irregular access patterns, thus guiding design improvements.

In short, benchmarks act as both reference workloads and performance standards that tie together otherwise fragmented research into a coherent, comparable discourse.

12.1. Types of Benchmarks

From a functional perspective, PIM benchmarks fall into two broad categories: *micro-level* or *architectural-level* benchmarks.

12.1.1. Micro-Level Benchmarks

These are fine-grained that evaluate small, isolated functions such as memory copy, vector addition, or arithmetic reduction. They are ideal for measuring core metrics like latency, bandwidth, and energy consumption of specific architectural components. They are lightweight and well-suited for validating correctness or testing specific architectural features; however, they do not reflect the complex interactions present in the end-to-end applications.

12.1.2. Architectural-Level Benchmarks

These benchmarks include complete workloads, synthetic traces, or high-level applications (e.g., neural networks, graph analytics). They evaluate the behavior of the entire system, including memory hierarchy, compute patterns, interconnect behavior, and scalability. As such, they are essential for assessing end-to-end performance and energy efficiency in realistic scenarios.

Benchmarks may also be classified by their origin and specialization into two categories; *general* and *PIM-specific* benchmarks.

12.1.3. General Benchmarks

General benchmarks are usually derived from traditional CPU or GPU suites, such as SPEC [97], Phoenix [98], Rodinia [99], or GAP [100]. These are partially modified to offload specific kernels onto PIM hardware. While they provide familiar workloads, they may not fully capture PIM-specific execution characteristics.

12.1.4. PIM-specific Benchmarks

Designed from the ground up to target PIM architectures, these benchmarks emphasize memory-centric computations, including bitwise logic, near-bank reductions, or matrix–vector operations executed directly within memory arrays. These workloads are essential for uncovering behaviors unique to PIM systems.

12.2. Simulator-Integrated Benchmarking Environments

Several simulators integrate built-in benchmark suites to support reproducible and low-level exploration of PIM behavior.

12.2.1. PIMSim and PiMulator

These simulators provide lightweight, self-contained workloads that users can execute directly for validating functionality or exploring core performance metrics [2, 101].

PIMSim includes a built-in suite of vector and logic kernels (e.g., addition, copy, bitwise operations) and a few graph-oriented routines like BFS and PageRank, based on the PEI benchmark set. **PiMulator** extends this concept by integrating the Hopscotch microbenchmark suite along with native PIM primitives such as RowClone, LISA, and Ambit. This enables users to experiment with near-bank and in-DRAM compute patterns.

A few simulators deliver more comprehensive benchmark collections that model real applications rather than synthetic kernels. **uPIMulator** is tightly coupled with the PrIM benchmark suite, providing ready-to-run workloads such as GEMV, SpMV, histogram, and database filtering directly within its framework [27]. **PIMeval**, on the other hand, supports the full **PIMbench** benchmark set, which includes 16 diverse tasks ranging from logical operations to machine-learning workloads, making it one of the most complete toolchains for evaluating digital DRAM PIM architectures [6]. **MNSIM** 2.0 incorporates end-to-end simulation of neural network workloads such as LeNet, VGG, and ResNet under varying quantization and process variation conditions [26]. **MemTorch** provides a modular interface for converting standard PyTorch models into memristive PIM workloads, allowing evaluation of both analog and digital compute-in-memory accelerators without additional integration effort [38].

Some simulators such as **Ramulator-PIM**, **MultiPIM**, and **MPU-Sim** rely entirely on external benchmark suites (e.g., Rodinia, GAP, CUDA kernels) used for validation studies [3, 7, 23]. While these demonstrate architectural feasibility, they lack standardized internal benchmarking environments, reducing reproducibility. As a result, only a subset of simulators provide integrated, reusable benchmarking environments that support consistent, in-depth exploration of PIM systems.

12.2.2. Dedicated Benchmark Suites for PIM

Recognizing the need for standardization, several benchmark suites have been developed specifically to support PIM research. The most prominent among them are **InSituBench**, **PrIM**, and **PIMbench**.

12.2.3. InSituBench Benchmark

InSituBench was proposed as part of the Fulcrum framework, which aims to simplify the design and evaluation of in-situ accelerators, systems that perform computation directly inside memory arrays [102]. The core

motivation was to address the inconsistency in workloads used by researchers, which made fair comparisons across PIM architectures difficult.

Unlike general-purpose CPU or GPU benchmarks, **InSituBench** focuses on the specific operational characteristics of in-memory and near-data processing, such as limited compute granularity, data locality, and reduced off-chip communication.

InSituBench contains a diverse mix of workloads chosen to represent the most common access and computation patterns seen in real PIM use cases. These workloads are grouped into five categories [102]:

1. *Bitwise Operations*: Basic logical primitives (AND, OR, XOR, NOT) used in PIM logic operations and bit-serial arithmetic.
2. *Arithmetic Kernels*: Integer and floating-point addition and multiplication operations, typical in reduction, histogramming, and filtering tasks.
3. *Data Movement and Reduction*: Includes memory copy, initialization, and aggregation kernels (sum, max), focusing on minimizing data transfer.
4. *Vector and Matrix Workloads*: GEMV, SpMV, and Conv2D kernels simulate ML and signal processing tasks.
5. *Graph and Search Tasks*: BFS, PageRank, and histogram traversal, designed to stress irregular memory access and fine-grained control.

Each workload operates on an abstract model built around the Fulcrum’s dispatch mechanism, which defines a simple yet flexible interface for dispatching in-situ operations within a memory stack. This enables the same benchmarks to be executed under different architectural configurations allowing fair comparisons across platforms. The authors validate the suite on both FPGA-based prototypes and cycle-level simulators, demonstrating how it captures the trade-offs between flexibility, parallelism, and control overhead in real in-situ architectures [102].

In summary, **InSituBench** establishes a comprehensive, PIM-specific benchmarking framework covering low-level bitwise operations, arithmetic computation, data-movement primitives, and higher-level analytical kernels. Its modular structure allows designers to isolate the effects of hardware organization, communication hierarchy, and control granularity. By bridging low-level kernel evaluation with application-inspired workloads, **InSituBench** provides a practical and reproducible foundation for comparing emerging in-situ and PIM architectures.

12.2.4. PrIM

Processing-In-Memory benchmarks(PrIM) is the first comprehensive benchmark suite designed specifically for evaluating commercial PIM hardware, particularly the UPMEM’s architecture [103]. PrIM provides an open-source set of workloads that cover a wide range of memory-bound applications. Unlike synthetic microbenchmarks or simulation-only workloads, PrIM enables researchers to study real performance, scalability, and energy efficiency compared to CPUs and GPUs.

The benchmark suite includes 16 workloads, each selected from diverse domains such as dense and sparse linear algebra, databases, analytics, graph processing, neural networks, bioinformatics, and image processing [103]. The workloads are categorized as follows:

1. *Linear Algebra*: GEMV, SpMV, and Vector Addition.
2. *Databases*: Filtering and uniqueness checks.
3. *Analytics*: Binary search and time-series analysis.
4. *Graph Algorithms*: BFS and traversal tasks.
5. *Neural Networks*: MLP inference under constrained memory.
6. *Bioinformatics*: Needleman–Wunsch alignment.
7. *Image Processing*: Histogram kernels.
8. *Parallel Primitives*: Reduction, scan, and transpose operations.

PrIM’s key strength lies in its well-balanced combination of workload diversity and practical relevance. The suite encompasses a broad spectrum of applications, ranging from simple data-parallel patterns such as vector addition and reduction to complex, synchronization-intensive tasks like BFS and Needleman–Wunsch (NW). This extensive coverage effectively reflects the complete computational range encountered in memory-bound workloads. Each benchmark within PrIM has been rigorously characterized using the roofline performance model, confirming that performance is genuinely constrained by memory access bandwidth rather than by compute throughput [103].

The authors evaluated PrIM on two different UPMEM PIM systems—one with 640 DPU s and another with 2,556 DPU s—and compared the results against those from state-of-the-art CPU and GPU platforms. Their findings demonstrated that PIM architectures can achieve substantial performance improvements, with speedups of up to 23 \times and energy efficiency gains of approximately 5 \times . These benefits are particularly pronounced for workloads

that are heavily memory-bound, involve relatively simple arithmetic operations, and exhibit minimal inter-DPU communication overhead [103]. In essence, **PrIM** serves a dual purpose: it functions both as a rigorous scientific benchmark suite and as a diagnostic tool that identifies which classes of workloads are most amenable to PIM acceleration. Additionally, it provides valuable guidance for co-design efforts in both hardware and software aimed at optimizing future memory-centric computing architectures.

12.2.5. *PIMbench*

PIMbench is part of the **PIMeval/PIMbench** framework, a unified platform designed for modeling, simulating, and benchmarking of digital DRAM-based PIM architectures [6]. This framework was developed to address a significant gap in the PIM research community: the lack of a portable and standardized benchmark suite applicable across wide variety of PIM designs including bit-serial, subarray-level, and bank-level architectures. Unlike suites benchmarks such as **PrIM** or **InSituBench**, which are tied to specific hardware implementations, **PIMbench** provides a high-level PIM programming API that allows the same benchmark code to execute unmodified on different architectural platforms, promoting reproducibility and enabling fair cross-platform comparison.

PIMbench contains 16 benchmarks covering a wide range of computation and data access patterns, categorized into multiple domains [6]:

1. *Linear Algebra*: includes Vector Addition, AXPY, GEMV, and GEMM designed to evaluate streaming memory access and arithmetic throughput.
2. *Sorting and Cryptography*: Radix Sort evaluates combined PIM–host execution performance, while AES encryption and decryption benchmarks measure logical intensity and bitwise computation.
3. *Graph and Database*: Triangle Count and Filter-by-Key benchmarks evaluate random access and associative processing capabilities.
4. *Image Processing*: includes Histogram, Brightness Adjustment, and Image Downsampling, which capture data aggregation and pixel-level computation.
5. *Machine Learning*: K-Nearest Neighbors (KNN), Linear Regression, K-means clustering, and three convolutional neural networks (VGG-13, VGG-16, VGG-19) model both supervised and unsupervised learning workloads under varying arithmetic and communication demands.

Each workload specifies memory access pattern (sequential vs. random) and execution type (PIM-only or PIM + Host), thereby emphasizing how data layout and host interaction affect overall system performance. Collectively, these benchmarks span from simple computational kernels (like addition or reduction) to more complex hybrid tasks that combine in-memory and host-side execution [6].

PIMeval simulator is evaluated across three digital DRAM-based PIM architectures: (1) subarray-level bit-serial, (2) Fulcrum-style bit-parallel, and (3) bank-level PIM. Their results show that bit-serial PIM excels at massively parallel logical and reduction operations, Fulcrum’s bit-parallel design performs best on arithmetic-heavy kernels, and bank-level PIM is advantageous for workloads requiring broader data access [6]. Overall, **PIMbench** facilitates quantitative, cross-architecture, quantitative evaluation of performance, energy efficiency, and scalability. In summary, **PIMbench** represents a major advancement toward establishing a standardized benchmarking methodology within the PIM community. By combining a diverse range of workloads with a portable API interface, it effectively bridges the gap between architectural simulation and system-level analysis, promoting reproducibility and accelerating research in memory-centric computing.

13. Comparative Study of PIM Simulators

To better understand the capabilities and limitations of current Processing-in-Memory (PIM) simulation tools, we performed a comparative study using two representative open-source simulators: **PIMeval** [39] and **PIMSimulator** [4]. Both tools aim to provide quantitative insights into PIM architectures, but they differ substantially in modeling depth, configuration flexibility, and output granularity. We follow two goals in this study:

- Evaluating how well these frameworks model the performance of common key workloads.
- Understanding the modeling assumptions and simulation methodologies behind each tool.

13.1. Motivation and workload selection

Matrix multiplication kernels form the computational backbone of nearly all modern AI workloads. In both the training and inference phases of deep neural networks, large-scale linear algebra operations, particularly General

Matrix–Matrix Multiplication (GEMM) and General Matrix–Vector Multiplication (GEMV), dominate runtime, memory bandwidth consumption, and overall energy cost.

For instance, in Convolutional Neural Networks (CNNs), convolution operations are typically transformed into GEMM kernels through tensor reshaping, making matrix–matrix multiplication the central compute primitive. In Transformer-based architectures, GEMM operations appear extensively in both the multi-head attention projections (query, key, and value computations) and the feed-forward layers, while GEMV arises in autoregressive decoding or token-by-token inference, where a single embedding vector is multiplied by large model weight matrices.

As Processing-in-Memory (PIM) architectures are explicitly designed to minimize data movement and accelerate such dense linear algebra operations, GEMM and GEMV represent the most meaningful and representative workloads for evaluating PIM simulators.

We selected four representative workloads that vary both in data dimensionality and operation type:

1. 512×512 in 512×512 matrix–matrix multiplication
2. 1024×1024 in 1024×1024 matrix–matrix multiplication
3. 512×512 in 512×1 matrix–vector multiplication
4. 1024×1024 in 1024×1 matrix–vector multiplication

Together, these workloads capture both the compute- and memory-intensive ends of AI workloads, allowing us to evaluate how each simulator models parallelism, bandwidth utilization, and latency scaling.

13.2. Evaluated simulators

13.2.1. PIMeval/PIMbench

PIMeval is a detailed simulation and benchmarking framework for PIM-based architectures. It models the internal structure of memory devices at the rank, bank, and subarray levels, while also capturing the hierarchical organization of PIM cores distributed across the DRAM.

The simulator focuses primarily on timing and bandwidth modeling. It estimates key performance characteristics such as row activation latency, read/write latency, column access time, and achievable bandwidth. These fine-grained models enable PIMeval to predict the latency and throughput of large-scale memory operations under various architectural configurations.

PIMeval supports a range of memory configurations, including DDR, HBM, LPDDR, and GDDR, as shown in Table 7. While the original implementation described in the base paper focused solely on DDR, additional memory technologies have been introduced in the public GitHub repository. However, not all of these are fully implemented—e.g., HBM currently lacks support for bit-serial execution.

Table 7: PIMeval configuration example

Parameter	Value
Number of Ranks	1
Banks per Rank	128
Subarrays per Bank	32
Rows per Subarray	8192
Columns per Subarray	8192
Number of PIM Cores	2048
Simulation Target	Bank-level PIM device

For our experiment, we use (BANK_LEVEL), FULCRUM, and bit-serial (BITSERIAL) architectures. In the bank-level architecture, operations are directly mapped to DRAM banks, which makes it simple, energy efficient, and fast. On the other hand, FULCRUM, the Fulcrum architecture features parallelism at the sub-array level inside memory and more flexible dataflow, and BITSERIAL is a fully digital bit-serial PIM execution style, often optimized for logical operations and faces high area and time overhead for arithmetic operations.

PIMeval reports the following output metrics:

- *Operation Latency (ms)*: total delay for read, write, compute, and data movement operations
- *Bandwidth Utilization (GB/s)*: achieved throughput compared to the theoretical peak
- *Energy per Operation (mJ)*: estimated dynamic and static energy consumption
- *Datamovement’s latency (ms) and energy (mJ)*: estimated delay and consumed energy during data transfers between memory and processing units

These outputs make PIMeval a timing-accurate, architecture-oriented simulator, ideal for exploring how DRAM-level organization affects PIM performance.

13.2.2. *PIMSimulator*

PIMSimulator provides a modular, instruction-driven simulation framework designed to capture the logical and functional behavior of PIM-enabled DRAMs (based on HBM interface). Rather than focusing solely on physical timing, it models the execution flow of PIM instructions—such as addition, multiplication, reduction, and data movement—across distributed PIM cores.

This simulator supports a wide range of architectural configurations, including:

- Number of channels, ranks, and banks
- PIM core allocation across subarrays
- Memory read/write latency models
- Host–PIM communication overhead
- Custom PIM instruction sets and scheduling policy

Example rows of PIMSimulator hardware configuration used in our experiments are shown in Table 8.

PIMSimulator’s outputs are more abstract than PIMeval’s, focusing on operation counts, execution time, and memory transactions. This makes it suitable for exploring algorithmic mapping strategies (e.g., distributing GEMM tiles across PIM cores) or evaluating different instruction scheduling schemes. However, PIMeval is more realistic and considers energy overhead.

13.3. *Results and Discussions*

Table 9 summarizes the execution performance for all configurations. PIMeval demonstrated significantly higher performance in the FULCRUM architecture compared to Bank-level, highlighting the efficiency of subarrays in facilitating parallelism. Bit-serial architecture is slower per arithmetic operation, which is why we experience throughput degradation in this fine-grained bit-level model.

For the 1024×1024 GEMM, PIMeval’s BANK LEVEL mode achieved 408.2 GOPS, compared to 251.5 GOPS in FULCRUM and 64.7 K GOPS

Table 8: PIMSimulator configuration example

Parameter	Value
Number of Bank Groups	4
Number of PIM Blocks	4
Device Width	8
Burst Length (BL)	8
Read Latency (RL)	19 ns
Row-to-Row Delay (tRRDL)	7 ns
Row-to-Column Delay for READ (tRCDRD)	19 ns
Write Recovery Time (tWR)	20 ns
Clock Cycle Time (tCK)	0.75 ns
Power-Down Current – Precharge Power-Down Mode (IDD2P)	34 mA
Core Voltage (Vddc)	1.2 V
Wordline Boost Voltage (Vpp)	2.5 V

in the BITSERIAL mode. The 512×512 GEMM followed the same trend, scaling roughly with matrix size. In GEMV, performance dropped sharply, since vector operations expose less parallelism; the BANK LEVEL mode achieved 0.21 GOPS for the 1024×1024 GEMV, compared to 0.05 GOPS for PIMSimulator.

PIMSimulator consistently reports lower execution time compared to PIMEval as shown in Figures 12 and 13, as it mainly focuses on optimizing scheduling and performance without considering energy efficiency.

In terms of energy, the Bank-Level configuration is consistently the most energy-efficient. For 1024×1024 GEMM, it consumes 569 mJ, compared to 3.8 J for FULCRUM and >1 kJ for Bit-Serial. A similar pattern appears in GEMV workloads, where Bit-Serial energy grows disproportionately with matrix size.

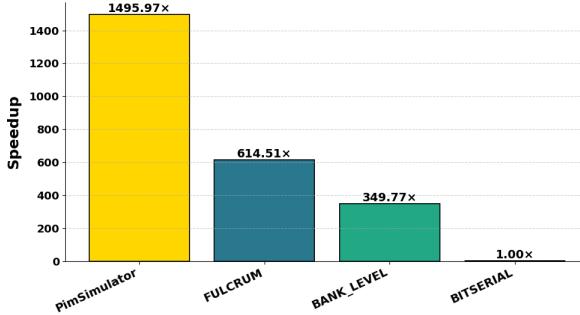


Figure 12: Speedup comparissson in PIMeval and PIMSimulator for 512×512 GEMV workload: PIMSimulator does not consider energy overhead, and totally focuses on optimizing latency and performance.

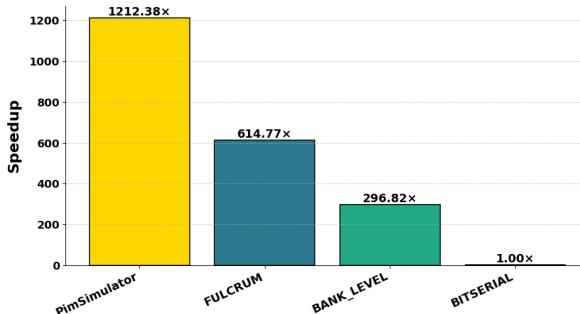


Figure 13: Speedup comparissson in PIMeval and PIMSimulator for 1024×1024 GEMV workload.

Table 9: Performance and Energy Comparison of PIMSimulator and PIMeval under GEMM/GEMV Workloads

Workload / Metric	PIMSimulator	PIMeval (Bank-Level)	PIMeval (FULCRUM)	PIMeval (BITSERIAL)
1024×1024 GEMM (ms)	–	408.23	251.49	64680.94
1024×1024 GEMV (ms)	0.0521	0.2128	0.1027	63.16
512×512 GEMM (ms)	–	88.31	62.88	16170.64
512×512 GEMV (ms)	0.0211	0.0903	0.0514	31.58
1024×1024 GEMM (mJ)	–	569.40	3831.23	1011350.62
1024×1024 GEMV (mJ)	–	0.3359	1.5006	987.65
512×512 GEMM (mJ)	–	129.09	957.99	237223.47
512×512 GEMV (mJ)	–	0.1524	0.7507	463.33

14. Conclusion

Processing-in-Memory (PIM) has matured from a conceptual paradigm into a central research direction for overcoming the memory wall in modern computing systems. By reducing data movement and integrating computation within memory structures, PIM offers a promising path toward higher energy efficiency and parallelism. However, this architectural shift also introduces unprecedented challenges in system modeling, workload characterization, and performance validation. Simulation has thus become the most indispensable tool in this evolution, providing the analytical foundation to explore, refine, and validate PIM concepts long before physical realization.

Over the past decade, simulation frameworks have evolved significantly, moving from simple memory latency models to sophisticated multi-level environments that capture detailed timing, architectural behavior, and application workloads. The diversity of simulators reflects the multidimensional nature of PIM itself: each framework balances accuracy, scalability, and flexibility in a distinct way. Some emphasize high-level abstraction to enable rapid exploration, while others offer cycle-accurate fidelity for microarchitectural verification. Together, these complementary approaches form an ecosystem that supports the entire design pipeline, from device characterization to algorithm evaluation.

A major outcome of this exploration is the growing emphasis on consistency and comparability. As PIM research expands across academic and industrial domains, standardized methodologies and benchmarks have become crucial for credible evaluation. The availability of open-source simulators and shared workloads has fostered reproducibility and accelerated progress, allowing diverse research groups to validate findings on common grounds. This trend highlights the collaborative nature of the PIM community and the pivotal role of simulation in sustaining it.

Looking ahead, the future of PIM simulation lies in hybrid and intelligent modeling. Adaptive frameworks that combine fast functional simulation with selective high-fidelity modeling can provide both scalability and precision. The integration of machine learning techniques promises to automate design-space exploration, enabling simulators to predict performance trends and optimize configurations dynamically. Moreover, tighter coupling between simulation and prototype hardware will bridge the gap between modeling and measurement, ensuring that simulation results remain grounded in physical reality.

Simulation is not merely a supporting tool but the driving engine of PIM innovation. It enables the translation of architectural imagination into verifiable insight and accelerates the path toward practical deployment. The continued refinement of simulation methodologies will determine how rapidly and reliably PIM technologies transition from research laboratories to mainstream computing systems.

References

- [1] M. Poremba, Y. Xie, Nvmain: An architectural-level main memory simulator for emerging non-volatile memories, in: 2012 IEEE Computer Society Annual Symposium on VLSI, IEEE, 2012, pp. 392–397.
- [2] S. Xu, X. Chen, Y. Wang, Y. Han, X. Qian, X. Li, Pimsim: A flexible and detailed processing-in-memory simulator, IEEE Computer Architecture Letters 18 (1) (2018) 6–9.
- [3] J. Gomez-Luna, G. d. Oliveira Junior, O. Mutlu, Ramulator-pim: A fast and flexible processing-in-memory simulation framework, accessed: 2025-10-09 (2025).
URL <https://github.com/CMU-SAFARI/ramulator-pim>
- [4] S. A. I. of Technology (SAIT), Pimsimulator: Cycle-accurate processing-in-memory simulator (commit 3703d1f), accessed: 2025-10-09 (2025).
URL <https://github.com/SAITPublic/PIMSimulator>
- [5] S. Mosanu, M. N. Sakib, T. Tracy, E. Cukurtas, A. Ahmed, P. Ivanov, S. Khan, K. Skadron, M. Stan, Pimulator: A fast and flexible processing-in-memory emulation platform, in: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2022, pp. 1473–1478.
- [6] F. A. Siddique, D. Guo, Z. Fan, M. Gholamrezaei, M. Baradaran, A. Ahmed, H. Abbot, K. Durrer, K. Nandagopal, E. Ermovick, K. Kiyawat, B. Gul, A. Mughrabi, A. Venkat, K. Skadron, Architectural modeling and benchmarking for digital dram pim, in: 2024 IEEE International Symposium on Workload Characterization (IISWC), 2024, pp. 247–261. doi:10.1109/IISWC63097.2024.00030.
- [7] C. Yu, S. Liu, S. Khan, Multipim: A detailed and configurable multi-stack processing-in-memory simulator, IEEE Computer Architecture Letters 20 (1) (2021) 54–57.
- [8] J. B. Buitrago Paniagua, Computer architecture simulation on gpu (2024).
URL <https://hdl.handle.net/10495/38543>

- [9] A. Akram, L. Sawalha, A survey of computer architecture simulation techniques and tools, *IEEE Access* 7 (2019) 78120–78145. doi:10.1109/ACCESS.2019.2917698.
- [10] A. Akram, L. Sawalha, $\times 86$ computer architecture simulators: A comparative study, in: 2016 IEEE 34th International Conference on Computer Design (ICCD), 2016, pp. 638–645. doi:10.1109/ICCD.2016.7753351.
- [11] L. Eeckhout, Computer Architecture Performance Evaluation Methods, 1st Edition, Morgan & Claypool Publishers, 2010.
- [12] J. T. Maurer, A. M. M. Ahmed, P. Khorrami, S. H. Moon, D. A. Reis, A survey on computing-in-memory (cim) and emerging nonvolatile memory (nvm) simulators, *Chips* 4 (2) (2025) 19.
- [13] A. Mohammadi, E. Cheshmikhani, H. Asadi, A reliability-aware replacement policy for stt-mram caches in server-class processors, *IEEE Transactions on Reliability* (2025).
- [14] E. Cheshmikhani, F. Shokouhinia, H. Farbeh, A low-cost fault-tolerant racetrack cache based on data compression, *IEEE Transactions on Circuits and Systems II: Express Briefs* 71 (8) (2024) 3940–3944.
- [15] D. Christ, L. Steiner, M. Jung, N. Wehn, Pimsys: A virtual prototype for processing in memory, in: Proceedings of the International Symposium on Memory Systems, MEMSYS '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 26–33. doi:10.1145/3695794.3695797.
URL <https://doi.org/10.1145/3695794.3695797>
- [16] J. Ahn, S. Hong, S. Yoo, O. Mutlu, K. Choi, A scalable processing-in-memory accelerator for parallel graph processing, in: Proceedings of the 42nd annual international symposium on computer architecture, 2015, pp. 105–117.
- [17] T. R. Kepe, E. C. de Almeida, M. A. Z. Alves, Database processing-in-memory: an experimental study, *Proc. VLDB Endow.* 13 (3) (2019) 334–347. doi:10.14778/3368289.3368298.
URL <https://doi.org/10.14778/3368289.3368298>

- [18] A. Denzler, G. F. Oliveira, N. Hajinazar, R. Bera, G. Singh, J. Gómez-Luna, O. Mutlu, Casper: Accelerating stencil computations using near-cache processing, *IEEE Access* 11 (2023) 22136–22154.
- [19] X. Dong, C. Xu, Y. Xie, N. P. Jouppi, Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31 (7) (2012) 994–1007.
- [20] E. Azarkhish, D. Rossi, I. Loi, L. Benini, Design and evaluation of a processing-in-memory architecture for the smart memory cube, in: *International Conference on Architecture of Computing Systems*, Springer, 2016, pp. 19–31.
- [21] G. F. Oliveira, O. Mutlu, Memory-centric computing systems tutorial (heart 2024), Tutorial handout, ETH Zurich, accessed: 2025-10-09 (2024).
URL <https://events.safari.ethz.ch/heart24-memorycentric-tutorial/lib/exe/fetch.php?file=handout.pdf>
- [22] G. Singh, J. Gómez-Luna, G. Mariani, G. F. Oliveira, S. Corda, S. Stuijk, O. Mutlu, H. Corporaal, Napel: Near-memory computing application performance prediction via ensemble learning, in: *Proceedings of the 56th annual design automation conference 2019*, 2019, pp. 1–6.
- [23] X. Xie, P. Gu, J. Huang, Y. Ding, Y. Xie, Mpu-sim: A simulator for in-dram near-bank processing architectures, *IEEE Computer Architecture Letters* 21 (1) (2021) 1–4.
- [24] A. Lu, X. Peng, W. Li, H. Jiang, S. Yu, Neurosim simulator for compute-in-memory hardware accelerator: Validation and benchmark, *Frontiers in artificial intelligence* 4 (2021) 659060.
- [25] C. Lammie, W. Xiang, B. Linares-Barranco, M. R. Azghadi, Memtorch: An open-source simulation framework for memristive deep learning systems, *Neurocomputing* 485 (2022) 124–133.
- [26] Z. Zhu, H. Sun, T. Xie, Y. Zhu, G. Dai, L. Xia, D. Niu, X. Chen, X. S. Hu, Y. Cao, et al., Mnsim 2.0: A behavior-level modeling tool for processing-in-memory architectures, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42 (11) (2023) 4112–4125.

- [27] B. Hyun, T. Kim, D. Lee, M. Rhu, Pathfinding future pim architectures by demystifying a commercial pim technology, in: 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA), IEEE, 2024, pp. 263–279.
- [28] C. Ortega, Y. Falevoz, R. Ayrignac, Pim-ai: A novel architecture for high-efficiency llm inference, arXiv preprint arXiv:2411.17309 (2024).
- [29] J. Vieira, N. Roma, G. Falcao, P. Tomás, Ndpmulator: Enabling full-system simulation for near-data accelerators from caches to dram, IEEE Access 12 (2024) 10349–10365.
- [30] H. Ham, J. Hong, G. Park, Y. Shin, O. Woo, W. Yang, J. Bae, E. Park, H. Sung, E. Lim, G. Kim, Low-overhead general-purpose near-data processing in cxl memory expanders, in: 2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO), 2024, pp. 594–611. doi:10.1109/MICRO61859.2024.00051.
- [31] X. Wang, Pimsim-nn: An isa-based simulation framework for processing-in-memory accelerators (commit 3e3442b), accessed: 2025-10-09 (2024).
URL <https://github.com/wangxy-2000/pimsim-nn>
- [32] J. Bai, F. Lu, K. Zhang, et al., Onnx: Open neural network exchange, <https://github.com/onnx/onnx> (2019).
- [33] Y. Gu, A. Khadem, S. Umesh, N. Liang, X. Servot, O. Mutlu, R. Iyer, R. Das, Pim is all you need: A cxl-enabled gpu-free system for large language model inference, in: Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, 2025, pp. 862–881.
- [34] J. Büchel, A. Vasilopoulos, W. A. Simon, I. Boybat, H. Tsai, G. W. Burr, H. Castro, B. Filipiak, M. Le Gallo, A. Rahimi, et al., Efficient scaling of large language models with mixture of experts and 3d analog in-memory computing, Nature Computational Science 5 (1) (2025) 13–26.
- [35] L. W. Nagel, D. Pederson, Spice (simulation program with integrated circuit emphasis), Tech. Rep. UCB/ERL M382 (Apr 1973).
URL <http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html>

- [36] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, V. Srinivas, Cacti 7: New tools for interconnect exploration in innovative off-chip memories, ACM Trans. Archit. Code Optim. 14 (2) (Jun. 2017). doi:10.1145/3085572.
 URL <https://doi.org/10.1145/3085572>
- [37] X. Peng, S. Huang, Dnn+neurosim v2.1: Benchmark framework for compute-in-memory accelerators for deep neural networks (on-chip training focus), commit 18289cb; Version announced: 2020-08-08; Accessed: 2025-10-09 (2024).
 URL https://github.com/neurosim/DNN_NeuroSim_V2.1
- [38] C. Lammie, V. Yon, J. Eshraghian, N. Garg, P. Drolet, Memtorch: Simulation framework for memristive deep learning systems (version 1.1.6), dOI of software release: 10.5281/zenodo.6279274; Accessed: 2025-10-09 (2022).
 URL <https://github.com/coreylammie/MemTorch>
- [39] F. A. Siddique, D. Guo, Z. Fan, M. Gholamrezaei, M. Baradaran, A. Ahmed, H. Abbot, K. Durrer, K. Nandagopal, E. Ermovick, K. Kiyawat, B. Gul, A. Mughrabi, A. Venkat, K. Skadron, Pimeval & pimbench: Performance and energy modeling framework for dram-based pim (commit 40ce162), accessed: 2025-10-09 (2025).
 URL <https://github.com/UVA-LavaLab/PIMeval-PIMbench>
- [40] S. Xu, X. Chen, Y. Han, X. Qian, Pimsim: Process-in-memory simulator (gem5 full-system compatible), commit 67c0fb6; Accessed: 2025-10-09 (2022).
 URL <https://github.com/vineodd/PIMSim>
- [41] Y. Kim, W. Yang, O. Mutlu, Ramulator: A fast and extensible dram simulator, IEEE Computer Architecture Letters 15 (1) (2016) 45–49. doi:10.1109/LCA.2015.2414456.
- [42] P. Rosenfeld, E. Cooper-Balis, B. Jacob, Dramsim2: A cycle accurate memory system simulator, IEEE Computer Architecture Letters 10 (1) (2011) 16–19. doi:10.1109/L-CA.2011.4.
- [43] S. Li, Z. Yang, D. Reddy, A. Srivastava, B. Jacob, Dramsim3: A cycle-accurate, thermal-capable dram simulator, IEEE Computer Architecture Letters 19 (2) (2020) 106–109. doi:10.1109/LCA.2020.2973991.

- [44] J. D. Leidel, Y. Chen, Hmc-sim-2.0: A simulation platform for exploring custom memory cube operations, in: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2016, pp. 621–630. doi:10.1109/IPDPSW.2016.43.
- [45] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, W. J. Dally, A detailed and flexible cycle-accurate network-on-chip simulator, in: 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2013, pp. 86–96. doi:10.1109/ISPASS.2013.6557149.
- [46] J. Cho, M. Kim, H. Choi, G. Heo, J. Park, LLMervingSim: A HW/SW Co-Simulation Infrastructure for LLM Inference Serving at Scale , in: 2024 IEEE International Symposium on Workload Characterization (IISWC), IEEE Computer Society, Los Alamitos, CA, USA, 2024, pp. 15–29. doi:10.1109/IISWC63097.2024.00012.
URL <https://doi.ieee.org/10.1109/IISWC63097.2024.00012>
- [47] W. Won, T. Heo, S. Rashidi, S. Sridharan, S. Srinivasan, T. Krishna, Astra-sim2. 0: Modeling hierarchical networks and disaggregated systems for large-model training at scale, in: 2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), IEEE, 2023, pp. 283–294.
- [48] Q. Zheng, X. Li, Y. Guan, Z. Wang, Y. Cai, Y. Chen, G. Sun, R. Huang, Pimulator-nn: An event-driven, cross-level simulation framework for processing-in-memory-based neural network accelerators, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 41 (12) (2022) 5464–5475.
- [49] M. Rosenblum, E. Bugnion, S. Devine, S. A. Herrod, Using the simos machine simulator to study complex computer systems, ACM Trans. Model. Comput. Simul. 7 (1) (1997) 78–103. doi:10.1145/244804.244807.
URL <https://doi.org/10.1145/244804.244807>
- [50] I. Fernández-Vega, R. Quislant-del Barrio, C. Giannoula, M. Alser, J. Gómez-Luna, E. D. Gutiérrez-Carrasco, Ó. G. Plata-González, O. Mutlu, et al., Natsa: A near-data processing accelerator for time series analysis. (2020).

- [51] M. Poremba, Nvmain: An architectural-level main memory simulator for emerging non-volatile memories (commit ad28c0c), accessed: 2025-10-09 (2018).
URL <https://github.com/SEAL-UCSB/NVmain>
- [52] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, K. Hsieh, K. T. Malladi, H. Zheng, O. Mutlu, Lazypim: An efficient cache coherence mechanism for processing-in-memory, *IEEE Computer Architecture Letters* 16 (1) (2017) 46–50. doi:10.1109/LCA.2016.2577557.
- [53] C. Ortega, S. Brocard, Upmem llm framework: Profiling and simulation for pytorch layers and functions, tag 8c64e9c; Accessed: 2025-10-09 (2024).
URL https://github.com/upmem/upmem_llm_framework
- [54] S. Chandrasekaran, C. Lin, S. Pasricha, Pim-enabled deep neural network acceleration: A survey, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36 (5) (2017) 817–829. doi:10.1109/TCAD.2016.2637799.
- [55] C. Bertolli, Y. Ge, Y. Chen, E. Ipek, Near-data processing: A modular approach for architecture-level simulation, in: Proceedings of the International Symposium on Computer Architecture (ISCA), 2016, pp. 123–134. doi:10.1145/3079856.3080237.
- [56] J. Park, H. Kim, S. Lee, Cycle-accurate architecture-level simulation of processing-in-memory systems, *Journal of Systems Architecture* 98 (2019) 210–221. doi:10.1016/j.sysarc.2019.01.005.
- [57] H. Li, Y. Zhang, L. Wang, Circuit-level modeling of emerging memory devices for pim architectures, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28 (6) (2020) 1471–1483. doi:10.1109/TVLSI.2020.2987890.
- [58] A. Dorostkar, H. Farbeh, H. R. Zarandi, An empirical fault vulnerability exploration of reram-based process-in-memory cnn accelerators, *IEEE Transactions on Reliability* 74 (1) (2024) 2290–2304.
- [59] Q. Zhang, Y. Liu, X. Chen, Device-level simulation of resistive ram for processing-in-memory applications, *IEEE Transactions on Electron Devices* 68 (3) (2021) 1234–1241. doi:10.1109/TED.2021.3056723.

- [60] A. A. Khan, J. P. C. D. Lima, H. Farzaneh, J. Castrillon, The landscape of compute-near-memory and compute-in-memory: A research and commercial overview (2024). arXiv:2401.14428.
URL <https://arxiv.org/abs/2401.14428>
- [61] D. Fujiki, X. Wang, A. Subramaniyan, R. Das, In-/near-memory computing, *Synthesis Lectures on Computer Architecture* 16 (2) (2021) 1–140.
- [62] K. Asifuzzaman, N. R. Miniskar, A. R. Young, F. Liu, J. S. Vetter, A survey on processing-in-memory techniques: Advances and challenges, *Memories-Materials, Devices, Circuits and Systems* 4 (2023) 100022.
- [63] Integrating and operating hbm2e memory, Micron technical brief, Micron Technology, Inc., rev. A (Apr. 2021).
URL <https://www.micron.com/products/ultra-bandwidth-solutions/hbm2e>
- [64] F. Devaux, The true processing in memory accelerator, in: 2019 IEEE Hot Chips 31 Symposium (HCS), 2019, pp. 1–24. doi:10.1109/HOTCHIPS.2019.8875680.
- [65] A. Alonso-Marín, I. Fernandez, Q. Aguado-Puig, J. Gómez-Luna, S. Marco-Sola, O. Mutlu, M. Moreto, Bimsa: accelerating long sequence alignment using processing-in-memory, *Bioinformatics* 40 (11) (2024) btae631.
- [66] B. Jahannia, S. A. Ghasemi, H. Farbeh, Multi-retention stt-mram architectures for iot: Evaluating the impact of retention levels and memory mapping schemes, *IEEE Access* 12 (2024) 26562–26580.
- [67] E. Cheshmikhani, H. Farbeh, H. Asadi, Robin: Incremental oblique interleaved ecc for reliability improvement in stt-mram caches, in: Proceedings of the 24th Asia and South Pacific Design Automation Conference, 2019, pp. 173–178.
- [68] D. Apalkov, B. Dieny, J. M. Slaughter, Magnetoresistive random access memory, *Proceedings of the IEEE* 104 (10) (2016) 1796–1830. doi:10.1109/JPROC.2016.2590142.
- [69] H. Farbeh, H. Kim, S. G. Miremadi, S. Kim, Floating-ecc: Dynamic repositioning of error correcting code bits for extending the lifetime

- of stt-ram caches, *IEEE Transactions on Computers* 65 (12) (2016) 3661–3675.
- [70] H. Li, T. F. Wu, S. Mitra, H.-S. P. Wong, Resistive ram-centric computing: Design and modeling methodology, *IEEE Transactions on Circuits and Systems I: Regular Papers* 64 (9) (2017) 2263–2273. doi:10.1109/TCSI.2017.2709812.
- [71] S. A. Ghasemi, B. Jahannia, H. Farbeh, Grapha: An efficient reram-based architecture to accelerate large scale graph processing, *Journal of Systems Architecture* 133 (2022) 102755.
- [72] G. W. Burr, M. J. Breitwisch, M. Franceschini, D. Garetto, K. Gopalakrishnan, B. Jackson, B. Kurdi, C. Lam, L. A. Lastras, A. Padilla, B. Rajendran, S. Raoux, R. S. Shenoy, Phase change memory technology, *Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena* 28 (2) (2010) 223–262. doi:10.1116/1.3301579.
URL <http://dx.doi.org/10.1116/1.3301579>
- [73] D. Reis, M. Niemier, X. S. Hu, Computing in memory with fefets, in: Proceedings of the international symposium on low power electronics and design, 2018, pp. 1–6.
- [74] X. Wang, X. Sun, Y. Han, X. Chen, Pimsim-nn: An isa-based simulation framework for processing-in-memory accelerators, in: 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2024, pp. 1–2. doi:10.23919/DATEx58400.2024.10546788.
- [75] I. Research, 3d-cim-llm inference simulator: Simulator for llm inference on an abstract 3d aimc-based accelerator, accessed: 2025-10-09 (2025).
URL <https://github.com/IBM/3D-CiM-LLM-Inference-Simulator>
- [76] H. A. Maruf, H. Wang, A. Dhanotia, J. Weiner, N. Agarwal, P. Bhattacharya, C. Petersen, M. Chowdhury, S. Kanaujia, P. Chauhan, Tpp: Transparent page placement for cxl-enabled tiered-memory, in: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, 2023, pp. 742–755.

- [77] V. Sze, Y.-H. Chen, T.-J. Yang, J. S. Emer, Efficient processing of deep neural networks: A tutorial and survey, *Proceedings of the IEEE* 105 (12) (2017) 2295–2329.
- [78] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, et al., Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings, in: *Proceedings of the 50th annual international symposium on computer architecture*, 2023, pp. 1–14.
- [79] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, B.-G. Chun, Orca: A distributed serving system for {Transformer-Based} generative models, in: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.
- [80] J. Choquette, W. Gandhi, O. Giroux, N. Stam, R. Krashinsky, Nvidia a100 tensor core gpu: Performance and innovation, *IEEE Micro* 41 (2) (2021) 29–35.
- [81] M. J. Rasch, D. Moreira, F. Carta, J. Büchel, C. Lammie, C. Mackin, K. Tran, T. Gokmen, M. Le Gallo-Bourdeau, K. El Maghraoui, Ibm analog hardware acceleration kit (aihwkit), tag c30db65; Zenodo DOI: 10.5281/zenodo.15467066; Accessed: 2025-10-09 (2025).
URL <https://github.com/IBM/aihwkit>
- [82] S. Spiga, A. Sebastian, D. Querlioz, B. Rajendran, *Memristive Devices for Brain-Inspired Computing: From Materials, Devices, and Circuits to Applications-Computational Memory, Deep Learning, and Spiking Neural Networks*, Woodhead Publishing, 2020.
- [83] M. Zhang, Y. Zhuo, C. Wang, M. Gao, Y. Wu, K. Chen, C. Kozyrakis, X. Qian, Graphp: Reducing communication for pim-based graph processing with efficient data partition, in: *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2018, pp. 544–557.
- [84] J. D. Leidel, Y. Chen, Hmc-sim-2.0: A simulation platform for exploring custom memory cube operations, in: *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016, pp. 621–630. doi:10.1109/IPDPSW.2016.43.

- [85] G. Dai, T. Huang, Y. Chi, J. Zhao, G. Sun, Y. Liu, Y. Wang, Y. Xie, H. Yang, Graphh: A processing-in-memory architecture for large-scale graph processing, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38 (4) (2018) 640–653.
- [86] J. Ahn, S. Yoo, O. Mutlu, K. Choi, Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture, *ACM SIGARCH Computer Architecture News* 43 (3S) (2015) 336–348.
- [87] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, M. Ignatowski, Top-pim: Throughput-oriented programmable processing in memory, in: *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, 2014, pp. 85–98.
- [88] M. A. Ibrahim, S. Aga, Pimacolaba: Collaborative acceleration for fft on commercial processing-in-memory architectures, in: *Proceedings of the International Symposium on Memory Systems*, 2024, pp. 13–25.
- [89] T. Xie, Z. Zhu, B. Li, Y. He, C. Li, G. Sun, H. Yang, Y. Xie, Y. Wang, Unindp: A unified compilation and simulation tool for near dram processing architectures, in: *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2025, pp. 624–640.
- [90] G. F. Oliveira, A. Olgun, A. G. Yağlıkçı, F. N. Bostancı, J. Gómez-Luna, S. Ghose, O. Mutlu, Mimdram: An end-to-end processing-using-dram system for high-throughput, energy-efficient and programmer-transparent multiple-instruction multiple-data computing, in: *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2024, pp. 186–203.
- [91] J. Park, J. Choi, K. Kyung, M. J. Kim, Y. Kwon, N. S. Kim, J. H. Ahn, Attacc! unleashing the power of pim for batched transformer-based generative model inference, in: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Volume 2, 2024, pp. 103–119.
- [92] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, J. Park, Neupims: Npu-pim heterogeneous acceleration for batched llm inferencing, in: *Proceedings of the 29th ACM International Conference*

on Architectural Support for Programming Languages and Operating Systems, Volume 3, 2024, pp. 722–737.

- [93] B. Hyun, T. Kim, D. Lee, M. Rhu, upimulator: A flexible and scalable simulation framework for general-purpose processing-in-memory (pim) architectures (commit 870d916), accessed: 2025-10-09 (2025).
URL <https://github.com/VIA-Research/uPIMulator>
- [94] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, K. E. Goodson, Phase change memory, Proceedings of the IEEE 98 (12) (2010) 2201–2227.
- [95] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, V. Srikumar, Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars, ACM SIGARCH Computer Architecture News 44 (3) (2016) 14–26.
- [96] J.-H. Han, R. E. West, K. Skadron, M. R. Stan, Thermal simulation of processing-in-memory devices using hotspot 7.0, in: 2021 27th International Workshop on Thermal Investigations of ICs and Systems (THERMINIC), IEEE, 2021, pp. 1–5.
- [97] J. Bucek, K.-D. Lange, J. v. Kistowski, Spec cpu2017: Next-generation compute benchmark, in: Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, 2018, pp. 41–42.
- [98] C. Kozyrakis, an api and runtime environment for data processing with mapreduce for shared-memory multi-core & multiprocessor systems., accessed: 2025-10-14 (2016).
URL <https://github.com/kozyraki/phoenix>
- [99] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, K. Skadron, Rodinia: A benchmark suite for heterogeneous computing, in: 2009 IEEE international symposium on workload characterization (IISWC), Ieee, 2009, pp. 44–54.
- [100] S. Beamer, K. Asanović, D. Patterson, The gap benchmark suite, arXiv preprint arXiv:1508.03619 (2015).

- [101] S. Mosanu, T. Tracy II, E. Cukurtas, P. Ivanov, Pimulator: Processing in memory emulation (commit 0fdd801), accessed: 2025-10-09 (2022). URL <https://github.com/hplp/PiMulator>
- [102] M. Lenjani, P. Gonzalez, E. Sadredini, S. Li, Y. Xie, A. Akel, S. Eilert, M. R. Stan, K. Skadron, Fulcrum: A simplified control and access mechanism toward flexible and practical in-situ accelerators, in: 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), IEEE, 2020, pp. 556–569.
- [103] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, O. Mutlu, Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware, in: 2021 12th International Green and Sustainable Computing Conference (IGSC), IEEE, 2021, pp. 1–7.